View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Electronic Thesis and Dissertation Archive - Università di Pisa

UNIVERSITA' DEGLI STUDI DI PISA

FACOLTA' DI INGEGNERIA

Corso di Laurea Specialistica in Ingegneria Informatica

Tesi di Laurea Specialistica

Tuning di cache NUCA way-adaptable

Relatori:

Prof. Cosimo Antonio Prete
Ing. Pierfrancesco Foglia

Candidato:

Di Mari Dario

A.A. 2006/2007

... ai miei genitori

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

La memoria cache è una particolare memoria che sfrutta la localizzazione spaziale e temporale dei dati per fornire un miglioramento del tempo medio di accesso ai dati da parte delle unità di elaborazione. Le memorie cache rivestono un ruolo importante all'interno degli attuali microprocessori. Infatti per ottenere un miglioramento delle prestazioni, negli ultimi anni si è assistito a un continuo aumento delle dimensioni della cache. Col passaggio dalle tecnologie microelettroniche a quelle nanoelettroniche, però, si è avuto un aumento di potenza in condizioni statiche dovuto alle correnti di perdita (leakage) dei transistor. Poiché tale consumo di potenza è proporzionale al numero di dispositivi fisici (transistor) le cache di grandi dimensioni sono una delle maggiori fonti di consumo di potenza statica. Quindi diventa di fondamentale importanza l'impiego di tecniche per il risparmio energetico per le memorie cache, focalizzate sul consumo di potenza statica.

Inoltre negli ultimi anni il progresso delle tecnologie microelettroniche ha reso possibile l'aumento delle prestazioni dei microprocessori: la continua diminuzione della feature size impiegata nei processi costruttivi dei circuiti integrati, ha permesso l'incremento del numero di transistor presenti all'interno dei singoli chip.

L'aumento del numero di transistor e delle frequenze operative, però, non è stato accompagnato da un progresso tecnologico significativo per quanto riguarda i collegamenti fisici all'interno dei circuiti integrati; per questo il ritardo introdotto dai fili (*wire-delay*), rispetto al periodo di clock, è aumentato [2].

Al momento si realizzano cache per cui si ipotizzano tempi di latenza costanti senza considerare che, in una cache L2 a più vie, quelle più

lontane dal controller avranno in futuro dei ritardi di propagazione sempre più elevati. Alla luce di queste considerazioni un modello di cache che consideri tempi di accesso uniformi risulterà inadeguato.
Relativamente ai sottosistemi di memoria, per mitigare il problema dello *wire-delay* , sono state introdotte le cache NUCA (Non Uniform Cache Architecture). Le NUCA sono cache di secondo livello (le cache L2 on-chip rappresentano una componente sempre più determinante nell'architettura dei microprocessori attuali, in termini di prestazioni, consumi e occupazione in area di silicio) che si differenziano dalle cache tradizionali (UCA - Uniform Cache Architecture) perché prevedono una matrice di banchi accessibili in maniera indipendente. Ogni banco è caratterizzato da un tempo di accesso che è funzione della posizione del banco stesso rispetto al controller e i banchi con minore latenza sono quelli che si trovano in prossimità di esso.
Il lavoro di tesi si baserà sulle operazioni di tuning applicate alla tecnica *way-adapting* a cache L2 di tipo D-NUCA: tale tecnica consiste nell'accensione o nello spegnimento di porzioni della cache a tempo di esecuzione, in base alla località dei riferimenti delle singole applicazioni. La tesi comprenderà l'implementazione di un algoritmo di predizione differenziale, derivato da uno già esistente [7].
L'obiettivo della tesi è di ottenere un risparmio di energia, mantenendo le performance su buoni livelli.
Le simulazioni sono state implementate con simulatori della famiglia *sim-alpha*; le rispettive simulazioni analizzate e confrontate, in termini di IPC (istruzioni su cicli di esecuzione) e di consumo di potenza, su un insieme di benchmark appartenenti alle suite SPEC2000 e NPB.

L'intero documento sarà strutturato come segue:

- *Capitolo 1*: viene descritto, in modo generale, lo stato dell'arte riguardo le memorie cache UCA e NUCA;
- *Capitolo 2*: si procede con la descrizione degli algoritmi di tuning e delle tecniche di riconfigurazione applicate alle cache D-NUCA way-adaptable;
- *Capitolo 3*: si evidenziano gli strumenti usati durante il lavoro svolto;
- *Capitolo 4* : vengono presentati i risultati raggiunti.

Capitolo 1 - Stato dell'arte

Nel presente capitolo verrà resa una panoramica in generale sulle tecnologie cache attualmente esistenti in commercio. Prima verranno esaminate le memorie cache denominate UCA (Uniform Cache Access); in seguito le memorie cache di tipo NUCA (Non Uniform Cache Architecture), nelle due organizzazioni S-NUCA e D-NUCA.
Infine verrà analizzato in dettaglio il modello D-NUCA (Dynamic-NUCA) che, grazie ad un meccanismo di migrazione dei dati, è risultato il più efficiente nel limitare gli effetti del *wire-delay* sulle prestazioni. Nel descrivere queste architetture di memoria ci si atterrà al lavoro svolto da Kim *et al.* [1] dell'Università del Texas.
La conclusione del capitolo si occuperà della tecnica *way-adapting* e della definizione di una cache D-NUCA *way-adaptable*, indispensabili nel lavoro svolto in questa tesi.

1.1 Memorie cache UCA e ML-UCA

Le cosiddette UCA (Uniform Cache Access) altro non sono che le tradizionali memorie cache, formate da un unico banco e caratterizzate da una struttura a blocchi, in cui ogni blocco ha una dimensione (larghezza) che dipende dalla quantità di informazione trasferibile tra due livelli di memoria , tra due livelli di cache, tra RAM e cache. In quanto cache tradizionali, le UCA sono affette dai comuni problemi di

latenza media, visto che il tempo di accesso per un blocco dati è calcolato uniformemente per tutti i blocchi nell'intero banco e non si tiene conto della locazione fisica che, risultando più o meno vicina al controllore della cache, può contribuire a significativi risparmi di tempo in termini di accesso.

Le varianti cache ML-UCA sono invece suddivise in una matrice di banchi di memoria in modo da diminuirne la latenza di accesso complessiva, e hanno un tempo di accesso calcolato su quello del banco più lontano dalla porta di ingresso della cache. L'accesso ai banchi più vicini perciò ha una latenza maggiore di quella teorica possibile.

Inoltre queste soluzioni multi-livelli, che riescono ad evitare eventuali conflitti fra i vari accessi, e che permettono più accessi contemporanei, sono tecnologicamente più costose e meno performanti perché devono inevitabilmente aumentare l'area del chip. Poiché ogni livello di cache si attiene alla politica dell'inclusione (per coerenza di cache), altro inconveniente è la presenza di ridondanza di dati, che non permette un uso efficiente dello spazio utilizzato.

In figura 1.1 sono illustrate le due organizzazioni di cache descritte finora:

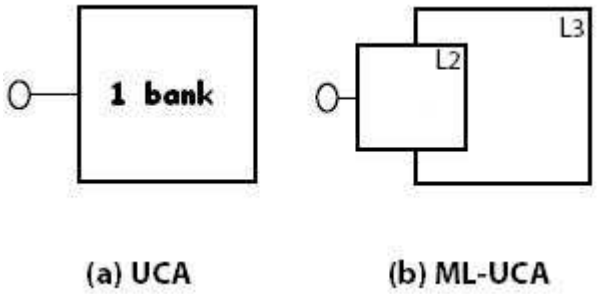


Figura 1.1 – Architetture cache tradizionali

In figura 1.1(a) è mostrato un unico banco di memoria di una cache UCA, mentre in figura 1.1(b) i due livelli di memoria L2 e L3 di una ML-UCA hanno un numero di banchi rispettivamente di 8 e 32.

Le cache in questione sono associative ad insiemi.
La seguente tabella riporta le prestazioni (in termini di IPC, miss rate, e latenze medie) di una cache UCA al variare della sua grandezza. Le cache provate sono da 2MB, 4MB, 8MB, e 16MB.

Tech. (nm)	L2 size (MB)	N° sub-banks	Unloaded latency	Loaded latency	IPC	Miss rate
130	2	16	13	67.7	0.41	0.23
100	4	16	18	91.1	0.39	0.20
70	8	32	26	144.2	0.34	0.17
50	16	32	41	255.1	0.26	0.13

Tabella 1.1 - Performance di una cache UCA

L'unloaded latency è il tempo medio di accesso (in cicli di clock) alla cache, assumendo di non essere in presenza di conflitti, a differenza del loaded latency che prevede anche i conflitti. Quest'ultima viene calcolata in modo sperimentale con l'ausilio di strumenti di simulazione.
La terza colonna indica la suddivisione in sotto-banchi di ogni banco della cache.
Si può notare come la latenza di accesso sia già alta in assenza di conflitti e come questi possano diventare un serio problema per le prestazioni (in media un accesso ad una cache da 16MB impiega 255 cicli di clock). Un conflitto può essere causato da due accessi diretti allo stesso banco (*bank contention*) oppure da due accessi che devono usare la stessa connessione (*channel contention*). Per ogni cache di grandezza diversa è inoltre riportato l'IPC trovato con il simulatore e la percentuale di miss in L2. Come si vede l'IPC decresce molto al crescere della dimensione della cache proprio perché cache di dimensioni maggiori sono più complesse (hanno un numero maggiore di banchi) e causano un innalzamento dei

ritardi complessivi. Si capisce quindi che questo tipo di architettura non sia applicabile per grandi cache o comunque per cache in cui la latenza dei collegamenti assuma un ruolo importante (*wire-dominated* cache).

1.2 Memorie cache NUCA

Organizzare una cache in cui ogni banco che la costituisce possiede una latenza propria, risolve i problemi di latenza visti con le soluzioni proprie della architettura UCA e ML-UCA (UCA multi livelli); infatti a differenza delle precedenti strutture di memoria dove vi erano latenze di accesso discrete[1], adesso con una memoria cache NUCA i banchi più vicini al controller della cache avranno una latenza di accesso minore rispetto agli stessi banchi posizionati più lontano, non pregiudicando il comportamento di tutti gli altri. Per facilitare ciò, c'è bisogno di una rete di interconnessione che permetta ad ogni banco un accesso indipendente.

Con questa architettura i dati sono mappati in modo statico nei banchi ed i bit più significativi della parte indice degli indirizzi determinano in quale banco avvenire l'accesso. Questo fa in modo che ogni banco sia indirizzabile indipendentemente dagli altri. La ricerca di un dato in una NUCA è incrementale, cioè una volta calcolato il giusto indice, viene confrontato il tag del banco della via più vicina al controller e solo in caso di miss si procede verso il banco successivo.

Il principale vantaggio di questi tipi di cache rispetto alle suddette cache multi-livello è che al loro interno non c'è la ripetizione dei blocchi da un livello ai successivi e ciò permette un notevole risparmio di spazio in quanto non è presente ridondanza dei dati nelle diverse vie.

[1] Nel caso di una ML-UCA a due livelli, la latenza di L3 può avere un valore che discosta molto da quello di L2.

È facile capire quindi che a parità di dimensione una cache NUCA può contenere più dati rispetto ad una equivalente UCA organizzata su più livelli.
Saranno ora presentate due particolari organizzazioni di cache NUCA con riguardo alla rete che connette internamente i banchi.

1.2.1 S-NUCA Private Channels

In questo modello di NUCA i banchi della cache vengono connessi al controller attraverso canali privati. Vi sarà quindi una propria connessione per i dati ed una per gli indirizzi, posta fra il controller della cache ed ogni banco. Il fatto che ogni banco può sfruttare la massima velocità possibile, in quanto non presenti conflitti di canale (channel contention), si traduce però in un grande overhead in termini di area di silicio occupata dal chip, che porta ad una limitazione nel numero di banchi costituenti la cache.

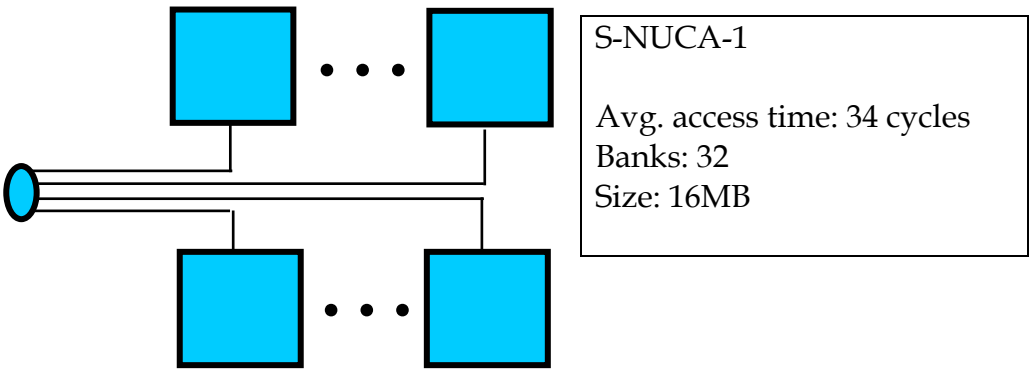


Figura 1.2 - Architettura S-NUCA-1

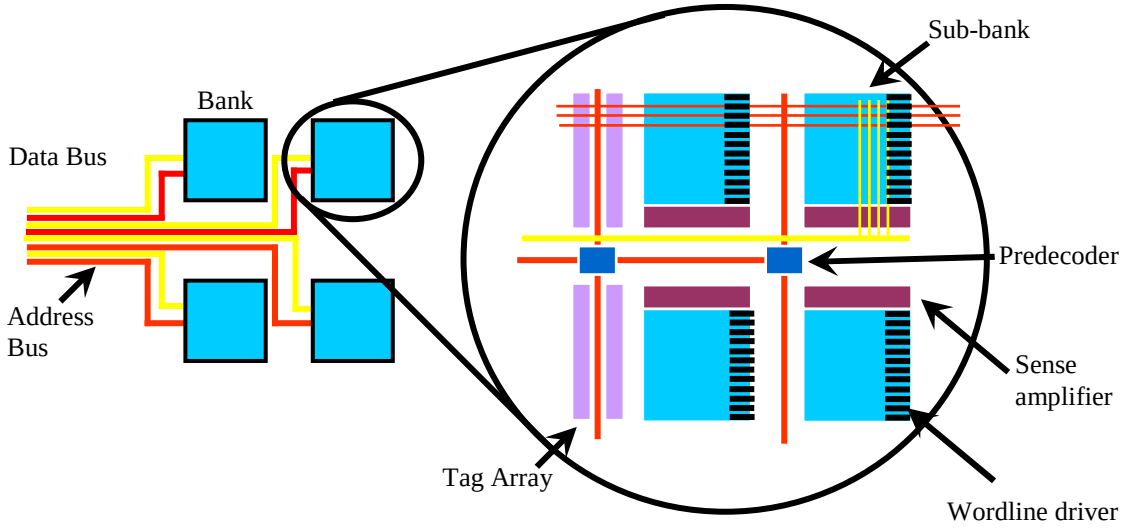


Figura 1.3 – Organizzazione interna di una S-NUCA-1

Come mostrato in figura 1.3 ogni banco di memoria è suddiviso in 4 sottobanchi e ciò spiega la presenza di un pre-decoder centrale, il quale ha il compito di smistare i vari segnali di accesso arrivati al banco.
La tabella successiva riassume le simulazioni fatte per diverse tecnologie costruttive della S-NUCA.

Tech.	**L2 size**	**N°**	**Unloaded latency**				**Conservative**		**Aggressive**	
(nm)	**(MB)**	**banks**	**bank**	**min**	**max**	**avg**	**Load.**	**IPC**	**Load.**	**IPC**
130	2	16	3	7	13	10	11.3	0.54	10.0	0.55
100	4	32	3	9	21	15	17.3	0.56	15.3	0.57
70	8	32	5	12	26	19	21.9	0.61	19.3	0.63
50	16	32	8	17	41	29	34.2	0.59	30.2	0.62

Tabella 1.2 – Performance di una cache S-NUCA-1

In tabella, per diverse dimensioni di cache, è riportato il numero di banchi che la costituisce (terza colonna), la latenza di accesso di ogni banco, il tempo di accesso minimo, corrispondente all'accesso al banco più vicino, il tempo di accesso massimo, corrispondente all'accesso al banco più lontano ed il tempo di accesso medio, cioè la media dei tempi di accesso di ogni banco (dalla quarta alla settima colonna).
Inoltre è riportato l'IPC e la latenza per due diversi approcci con e senza conflitti (conservativo e aggressivo rispettivamente).
La differenza sostanziale tra questi risultati e quelli ottenuti per le cache UCA è che in questo caso l'IPC cresce al crescere della dimensione e del numero di banchi della cache. Questo avviene perché all'aumentare del numero delle vie (e quindi della dimensione) non c'è un corrispondente aumento della latenza dei banchi delle vie precedenti, le quali rimangono indipendenti le une dalle altre. D'altro canto in questa soluzione il peso dei collegamenti privati si fa sentire per cache di grandi dimensioni, infatti la latenza massima, e quella media, della cache da 16MB subisce un innalzamento molto più alto rispetto alle cache più piccole. Questo spiega perché l'IPC della cache da 16MB si riduce del 2% rispetto a quello della cache da 8MB.
Infine il peso dei collegamenti privati si nota anche riguardo la suddivisione in banchi della cache: dalla terza colonna appare chiaro che il numero di banchi non va mai oltre 32. Questo avviene perché all'aumentare del numero dei banchi, i canali aumentano in numero ed in lunghezza, causando un aumento delle latenze di accesso ai singoli banchi.
Perciò, per ottenere cache di dimensioni maggiori si deve necessariamente aumentare la dimensione dei banchi, causando ancora una volta un aumento dei tempi di risposta dei banchi stessi.

1.2.2 S-NUCA Switched Channels

Il secondo modello di S-NUCA usa una connessione mesh con doppi canali che collegano semplici router situati presso ogni banco. I canali sono doppi in modo da consentire connessioni bidirezionali e limitare i conflitti. In questa implementazione c'è un ridotto uso di fili e per questo cresce il numero di banchi adottabili e anche la dimensione massima raggiungibile dalla cache. Si è sperimentato che sono le cache più grandi a beneficiare di questa implementazione. Diversamente il principale svantaggio di questa soluzione è dovuto alla maggiore complessità degli switch presso ogni banco.

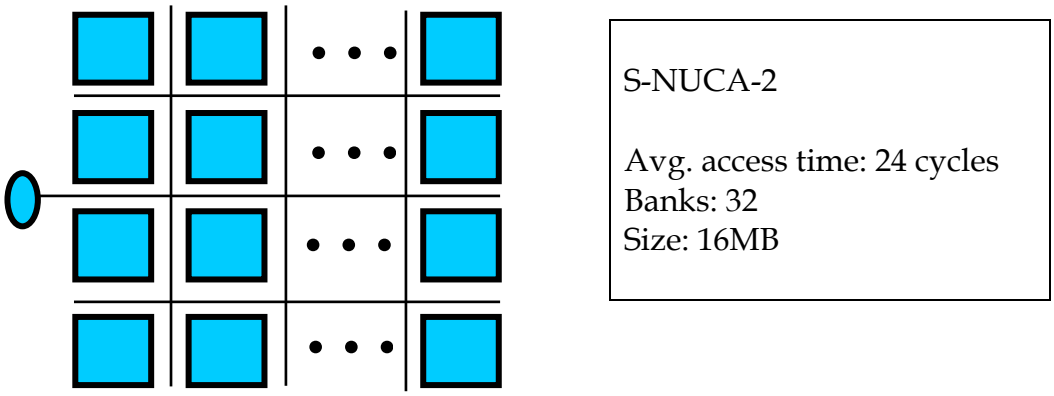


Figura 1.4 - Architettura S-NUCA-2

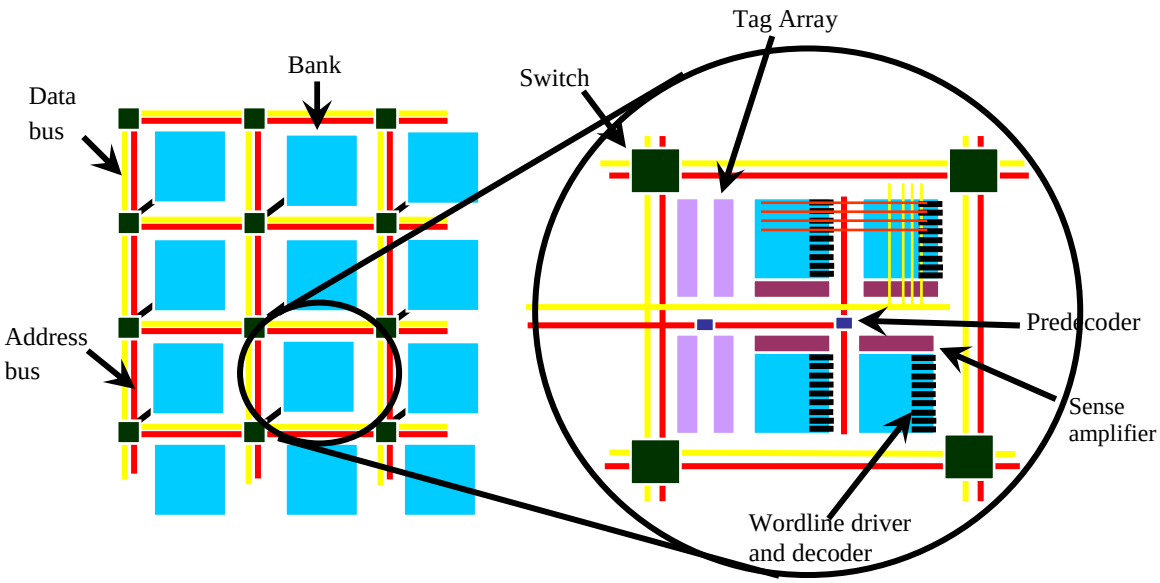


Figura 1.5 - Organizzazione interna di una S-NUCA-2

La figura 1.5 mostra chiaramente la rete mesh fra i banchi: ogni banco ha un proprio switch connesso agli switch dei banchi vicini.

L'estrema semplicità della rete permette una maggiore velocità negli accessi ai banchi; ciò viene chiarito nella tabella successiva dalle latenze riportate (min, max e avg).

La latenza del banco è il tempo di accesso dello stesso. I tempi min, max e avg indicano il ritardo di accesso ai banchi più vicini, a quelli più lontani ed il tempo medio. Sperimentalmente è calcolato l'IPC, la latenza in presenza di carico ed il numero di accessi ai banchi.

Tech. (nm)	L2 size (MB)	Num. banks	Unloaded latency				Loaded latency	IPC	Bank req.
			bank	min	max	avg			
130	2	16	3	4	11	8	9.7	0.55	17M
100	4	32	3	4	15	10	11.9	0.58	16M
70	8	32	5	6	29	18	20.6	0.62	15M
50	16	32	8	9	32	21	24.2	0.65	15M

Tabella 1.3 - Performance di una cache S-NUCA-2

1.3 Memorie cache D-NUCA

Mentre nelle S-NUCA i blocchi vengono associati ai rispettivi banchi in maniera statica, nelle D-NUCA tale associazione è variabile nel tempo. I risultati riportati da Kim *et al.* [1] dimostrano che le D-NUCA sono caratterizzate da una latenza media minore, concentrando i dati più utilizzati nei banchi in prossimità del controller; questa caratteristica le rende più performanti rispetto alle S-NUCA, e risulta fondamentale per l'introduzione di meccanismi per la riduzione del consumo di potenza.
Nelle memorie cache D-NUCA si introduce il meccanismo della migrazione dei blocchi per mantenere i dati acceduti più frequentemente nei banchi con minore latenza. Per questo è necessario prevedere una rete di collegamenti tra i banchi e introdurre degli switch per lo smistamento delle informazioni all'interno della rete stessa. La struttura della rete prevede uno switch per ogni banco; nella figura sottostante (la 1.6) è riportato lo schema architetturale di una cache D-NUCA.

Per l'implementazione di una cache D-NUCA occorre definire le seguenti politiche:

- *mapping*: come vengono associati i blocchi ai rispettivi banchi;
- *search*: come ricercare un blocco all'interno delle sue possibili locazioni;
- *promotion*: sotto quali condizioni i blocchi dovrebbero migrare da un banco all'altro;
- *insertion*: quale blocco rimpiazzare in caso di miss.

Nelle sezioni seguenti verranno descritte in dettaglio le suddette politiche.

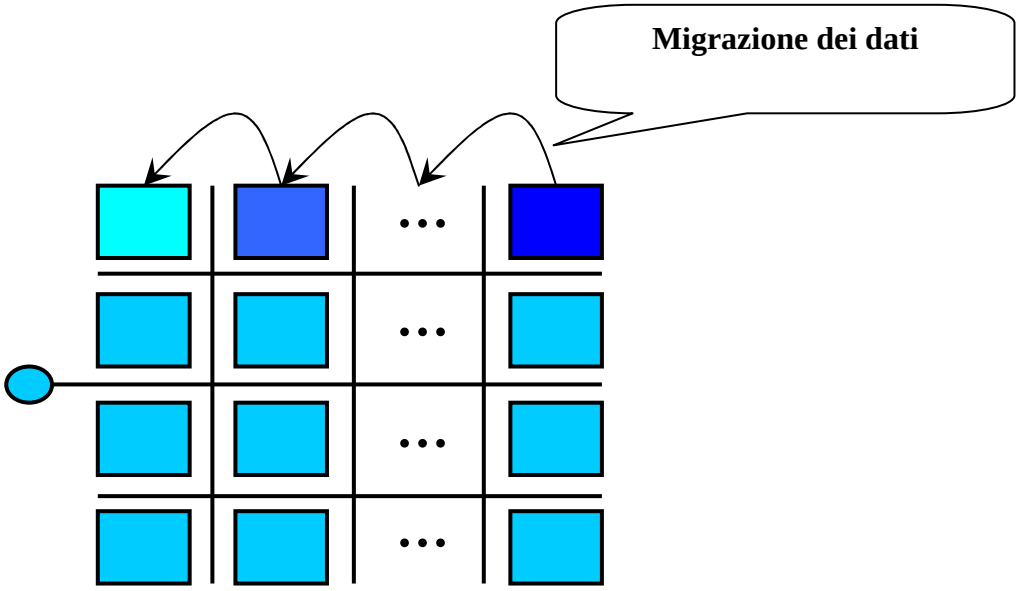


Figura 1.6 - Architettura D-NUCA

1.3.1 Mapping dei blocchi

Un blocco può essere mappato in un'unica posizione, come avviene nelle cache tradizionali di tipo *direct-mapped*; diversamente, la soluzione opposta prevede invece di mappare un blocco in un qualsiasi banco, in conformità alle cache di tipo *fully-associative*. Per le D-NUCA tale ultimo approccio risulta di difficile implementazione, quindi è stata avanzata una soluzione intermedia denominata spread sets, per cui l'intera cache è considerata come un'unica struttura set-associative: ogni set è distribuito su più banchi ed ogni banco contiene una singola via del set. L'insieme dei banchi usati per realizzare questo tipo di associatività prende il nome di bank set e il numero di banchi del bank set corrisponde all'associatività della cache. Per assegnare i banchi di una cache ai rispettivi bank set sono state indicate tre strategie: simple mapping, fair mapping e shared mapping.

Con la politica simple mapping ciascuna colonna di banchi rappresenta un bank set. Quindi un blocco viene individuato cercando per prima la colonna di banchi di appartenenza, cioè il bank set, quindi il relativo set, infine comparando i campi tag delle varie vie con l'indirizzo del blocco in questione. Questa politica risulta la più semplice da implementare anche se ha due inconvenienti: il primo è che l'associatività è dettata dall'organizzazione fisica della D-NUCA e può non coincidere con il valore ottimale; il secondo è che i set sono caratterizzati da latenze medie diverse, perché si trovano a distanze diverse dal controller. In figura 1.7a ogni riga di banchi assume il ruolo di bank set ed ogni banco che la compone appartiene ad una diversa via dello stesso insieme.

Con la politica fair mapping i banchi sono associati ai bank set in maniera che i tempi medi di accesso dei bank set stessi risultino simili. Lo svantaggio di questa soluzione sta principalmente nella maggiore complessità del percorso di routing da un banco all'altro all'interno di un bank set. In figura 1.7b è riportata la distribuzione dei bank set di una cache D-NUCA con 32 banchi, organizzati in 8 colonne e 4 righe: si tratta cioè di una cache 4-way *set-associative*.

Con la politica shared mapping i banchi più vicini al controller sono condivisi tra più bank set. Se ciascuno di questi banchi è condiviso tra n bank set, allora tutti i banchi della cache devono essere n-way *set-associative*, per far sì che lo spostamento di un blocco da un banco non condiviso ad un banco condiviso sia sempre possibile. In figura 1.7c viene considerata una politica shared mapping con ciascun banco di tipo 2-way *set-associative*.

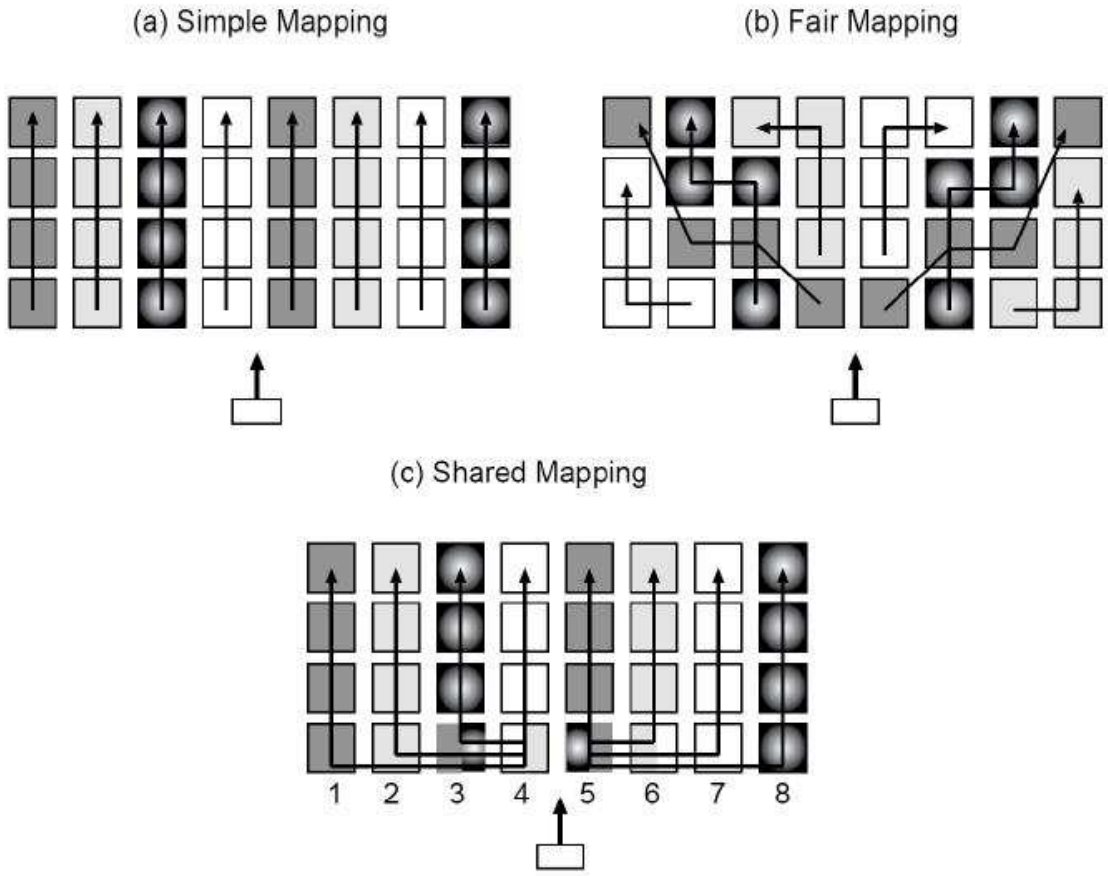


Figura 1.7 – Strategie di mapping

1.3.2 Search di un blocco

Una volta che a un indirizzo di blocco è stato associato un bank set occorre stabilire in quale via del set, e quindi in quale banco, tale blocco si trova. Poiché le D-NUCA prevedono un mapping dinamico questa operazione comporta la ricerca del blocco in ogni via, partendo dal banco più vicino al controller tra quelli componenti il bank set, fino a risolvere l'accesso in una hit o una miss. Per effettuare la ricerca di un blocco all'interno di un bank set sono state proposte due modalità principali. La prima modalità prende il nome di *incremental search* e

prevede che i banchi componenti il bank set siano acceduti sequenzialmente, partendo da quello più vicino al controller, finché il blocco viene individuato o si verifica una miss (che necessariamente si presenta in corrispondenza dell'accesso al banco più lontano). Questa soluzione riduce al minimo il traffico di rete e limita il consumo di potenza. L'altro modo, *multicast search*, permette accessi in parallelo ai banchi che costituiscono il bank set; il grado di parallelismo degli accessi è limitato dal ritardo di routing introdotto dalla rete. Tale soluzione permette un aumento delle prestazioni rispetto alla *incremental search* a scapito di un'intensificazione del traffico di rete ed un maggior consumo di potenza. Assieme a queste due modalità principali sono state proposte alcune varianti, come la *limited multicast* e la *partitioned multicast*, che sono una combinazione delle precedenti.

1.3.3 Promotion dei blocchi

Per concentrare gli accessi nei banchi più vicini al controller sarebbe auspicabile un ordinamento di tipo LRU (Least Recently Used) dei blocchi che compongono i vari set, con il primo dei banchi del bank set contenente il blocco MRU (Most Recently Used). Purtroppo una strategia simile porterebbe ad un numero eccessivo di trasferimenti di dati e ad un'eccessiva complessità circuitale, per cui la strategia che è stata proposta è soltanto un'approssimazione dell'ordinamento LRU. Tale strategia è stata denominata *generational promotion* e prevede che in caso di hit un blocco venga promosso nel banco che si trova nella posizione immediatamente precedente rispetto al controller, con relativo avvicendamento se la posizione è occupata da un blocco valido. La figura 1.8 mostra la distribuzione delle hit per l'algoritmo LRU e per il

generational promotion. Il confronto è stato effettuato tra una cache tradizionale 16-way set-associative e una D-NUCA con 16 banchi per bank set; i valori rappresentano una media sull'intera suite di benchmark utilizzati per le prove.

Fin qui non è stata fatta distinzione tra hit su operazioni di load e hit su operazioni di store; ovviamente la promotion può essere applicata indifferentemente in entrambi i casi. Comunque, nel lavoro di Kim *et al.* [1], la promotion viene attuata solo per le operazioni di load.

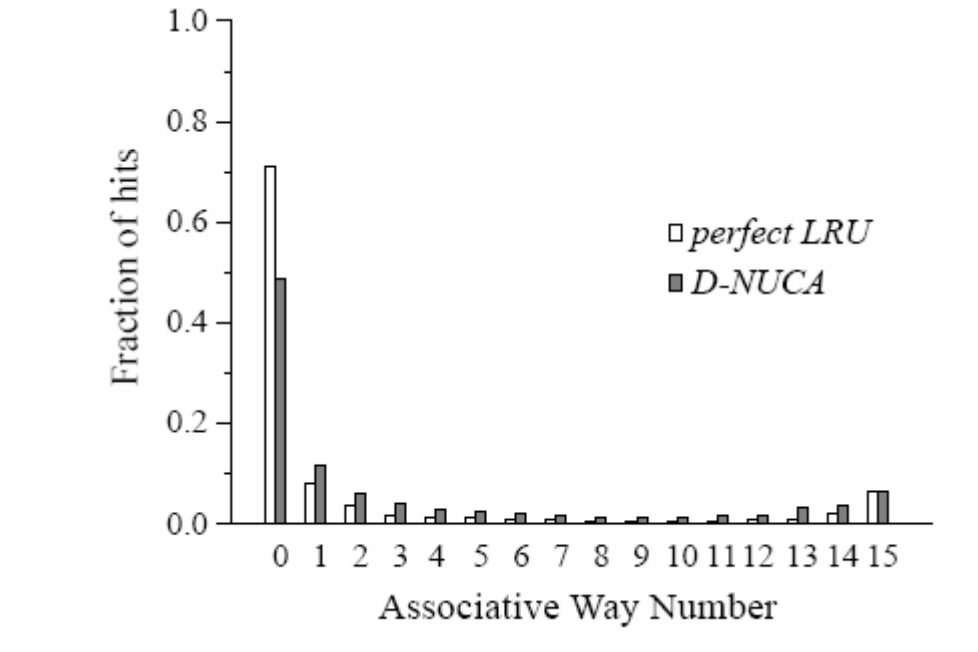


Figura 1.8 – Distribuzione delle hit per LRU e generational promotion

1.3.4 Insertion di un blocco

In caso di cache miss il blocco proveniente dal livello di memoria inferiore può essere inserito in un banco adiacente al controller o in uno più lontano. Kim *et al.* [1] hanno privilegiato la soluzione dell'inserimento in coda, cioè nel banco più lontano tra quelli componenti il bank set; con questa soluzione si evita che venga rimpiazzato un blocco che, trovandosi già vicino al controller, ha una grossa probabilità di essere riferito nuovamente.
Un'altra scelta riguarda le azioni da compiere sul blocco che viene rimpiazzato: la strategia di tipo zero-copy prevede che il blocco venga semplicemente eliminato dalla cache, mentre la strategia one-copy prevede che il blocco da rimpiazzare vada a sostituire un blocco in un banco a più bassa priorità. Alla politica di inserimento in coda è associata, necessariamente, la politica zero-copy.

1.3.5 Confronto prestazioni

In tabella 1.5 sono riportate le prestazioni IPC delle cache NUCA rispetto alle cache tradizionali (UCA), per diverse tecnologie costruttive.
Per quanto riguarda le NUCA, sono state prese in esame le configurazioni S-NUCA-1, S-NUCA-2 e D-NUCA. In particolare i dati delle D-NUCA si riferiscono ad una configurazione che prevede *simple mapping*, *multicast search*, *tail insertion* e *promotion* di tipo 1 bank/1 hit. La tabella 1.4 mostra le prestazioni in termini di latenze, IPC, e miss rate della D-NUCA in configurazione base. Nella terza colonna è riportata l'organizzazione della matrice di banchi di ogni dimensione di cache: la

cache da 2MB è composta da 4 bank sets ognuno di 4 vie, la cache da 4MB ha 8 bank sets da 4 vie, la cache da 8MB ha 16 bank sets da 8 vie ed infine la cache da 16MB ha 16 bank sets da 16 vie. Nel secondo blocco della tabella si può leggere il tempo di accesso di ogni banco, la latenza di accesso minima (latenza per l'accesso al banco più vicino), la latenza di accesso massima (latenza per l'accesso al banco più lontano) e la latenza media di accesso, tutte considerate in assenza di conflitti. Si nota che a differenza delle S-NUCA 1, le latenze massime sono estremamente basse considerato l'elevato numero di banchi che compone la cache. La cache da 16MB infatti è composta da ben 256 banchi, ma nonostante ciò ha una latenza massima di soli 47 cicli di clock (una S-NUCA 1 da 16MB è composta da soli 32 banchi ed ha una latenza massima di 41 cicli). Questo permette di poter aumentare più liberamente il numero di banchi riducendone la dimensione e quindi il tempo di accesso introdotto da ognuno di essi. Si nota infatti che il tempo di accesso dei banchi anche per le grandi cache è di soli 3 cicli, mentre per soluzioni S-NUCA (sia in versione 1 che 2), che presentano banchi di dimensioni maggiori, questo ritardo cresce fino a 8 cicli. Per quanto riguarda le prestazioni si osserva che per le cache di più piccole dimensioni (4MB) si ha un leggero miglioramento (circa il 2%) sulle prestazioni delle S-NUCA più performanti, mentre per dimensioni maggiori (16MB) il miglioramento può arrivare fino al 6%.

Lo svantaggio principale di queste cache riguarda l'elevato numero di messaggi che circolano all'interno della rete di interconnessione ed il conseguente alto numero di accessi ai banchi (266M contro 15M per cache da 16MB) anche in conseguenza del fatto che per questa cache si è scelta la ricerca in modalità *multicast*. Tutto ciò aumenta il consumo energetico del dispositivo ed aumenta la probabilità di conflitti fra due o più accessi contemporanei. Si deve considerare però che ogni banco è molto più piccolo rispetto al caso S-NUCA (quindi consumerà meno) e che il loro numero è molto più grande. È riportata l'organizzazione della

matrice di banchi (bank org.), le latenze in assenza di conflitti (unloaded latency), le latenze con carico, l'IPC, il miss rate, ed il numero di accessi ai banchi.

L'IPC è calcolato come media tra gli IPC dei singoli benchmark.

Tech. (nm)	**L2 size (MB)**	**Bank org.**	**Unloaded latency**				**IPC**	**Miss rate**	**Bank req.**
			bank	**min**	**max**	**avg**			
130	2	4x4	3	4	11	8	0.57	0.23	73M
100	4	8x4	3	4	15	10	0.63	0.19	72M
70	8	16x8	3	4	31	18	0.67	0.15	138M
50	16	16x16	3	3	47	25	0.71	0.11	266M

Tabella 1.4 – Performance di una cache D-NUCA

Type	Technology (nm)	L2 size (MB)	N° banks	IPC
UCA	130	2	1	0.41
UCA	100	4	1	0.39
UCA	70	8	1	0.34
UCA	50	16	1	0.26
S-NUCA-1	130	2	16	0.55
S-NUCA-1	100	4	32	0.57
S-NUCA-1	70	8	32	0.63
S-NUCA-1	50	16	32	0.62
S-NUCA-2	130	2	16	0.55
S-NUCA-2	100	4	32	0.58
S-NUCA-2	70	8	32	0.62
S-NUCA-2	50	16	32	0.65
D-NUCA	130	2	16	0.57
D-NUCA	100	4	32	0.63
D-NUCA	70	8	128	0.67
D-NUCA	50	16	256	0.71

Tabella 1.5 – Confronto IPC: UCA, S-NUCA-1, S-NUCA-2, D-NUCA

1.4 Tecnica way-adapting

Nell'articolo di Kobayashi *et al.* [3] viene descritto un meccanismo di riconfigurazione dinamica della cache che consente la riduzione del consumo di potenza utilizzando le informazioni sulla località dei riferimenti delle applicazioni. La riconfigurazione consiste nell'accensione o nello spegnimento delle singole vie della cache a tempo di esecuzione. Poiché la località dei riferimenti per ogni applicazione è diversa, il contributo che può dare ogni via di una cache *set-associative* alle performance, non solo varia utilizzando suite differenti di applicazioni, ma varia anche utilizzando applicazioni diverse all'interno di ogni stessa suite. Tutto ciò si traduce in un minor consumo di potenza (con una esigua perdita di performance), disabilitando le vie delle cache meno utilizzate dalle diverse applicazioni.

La tecnologia alla base dello way-adapting è la *Gated -*V_{dd} [6].

Qust'ultima introduce nelle celle SRAM un transistor ad alta V_t tra il circuito e un collegamento, quale può essere quello verso l'alimentazione o verso massa. Il transistor è un elemento di passo ed è comandato da un segnale di controllo: se il segnale è attivo il transistor è in conduzione e la cella SRAM si comporta normalmente, altrimenti la cella, essendo isolata dall'alimentazione, perde lo stato che aveva precedentemente memorizzato e si porta in una condizione per cui la potenza di *leakage* è praticamente nulla.

Per limitare l'aumento di area, il transistor di attivazione e il meccanismo di controllo possono essere condivisi tra tutte le celle di una *wordline* della SRAM; con questo accorgimento la complessità circuitale della soluzione è piuttosto ridotta. Lo svantaggio di questa tecnica è l'aumento di miss a causa della perdita del contenuto delle celle; per

ovviare a questo è possibile introdurre dei meccanismi che, utilizzando le informazioni sulla località degli accessi in memoria dei programmi, riescano a concentrare i dati più frequentemente acceduti in porzioni ben determinate della cache, così da applicare lo spegnimento a celle che hanno una bassa probabilità di essere nuovamente riferite.

In tabella 1.6 sono mostrati i risultati circa le località dei riferimenti per diverse suite: *SPECint/fp*, *MediaBench*, *anagram* e *dhrystone*. Per cache references si intende il numero di accessi in L1 instruction e data cache ordinati per il loro stato del campo LRU.

Suite	**Benchmark**	**Functions**	**Instruction executed**	**Cache references**
SPECint95	go	go program	548M	157M
	gcc	Based on the GNU C Compiler	281M	109M
	compress	UNIX Utility program	80M	28M
	perl	Programming language	21M	8M
SPECfp2000	ammp	Computational Chemistry	21M	8M
	art	Image Recognition / Neural Net.	1B	200M
	equake	Seismic Wave Propagation Simul.	1B	170M
MediaBench	mpeg2encode	Mpeg2 encoder	1B	146M
	mpeg2decode	Mpeg2 decoder	171M	32M
	g721-encode	Voice compression encoder	585M	270M
	g721-decode	Voice compression decoder	558M	256M
	rasta	Speech recognition	19M	6M
Misc	anagram	Puzzle	17M	6M
	dhrystone	Synthetic benchmark	526M	173M

Tabella 1.6 – Località dei benchmark a confronto

Nelle cache *way-adaptable* la località dei riferimenti viene analizzata a livello locale e a livello globale: a livello locale viene considerato lo stato del campo LRU delle entry che vengono accedute nell'arco di un certo periodo; a livello globale, invece, viene utilizzata una macchina a stati per regolare il meccanismo di accensione/spegnimento delle vie, determinante soprattutto in presenza di comportamenti irregolari della località.

Per quanto riguarda il comportamento locale, viene utilizzata una metrica basata sul parametro *D*, così definito:

$$D = LRU_{count} / MRU_{count} \tag{1.1}$$

dove LRU_{count} e MRU_{count} rappresentano rispettivamente il numero di riferimenti ai blocchi LRU e MRU dei vari set all'interno di un determinato periodo. Questa metrica non utilizza l'intero stato del campo LRU delle entry riferite per non introdurre eccessiva complessità a livello circuitale.

In figura 1.9 sono schematizzati due scenari diversi dal punto di vista della località.

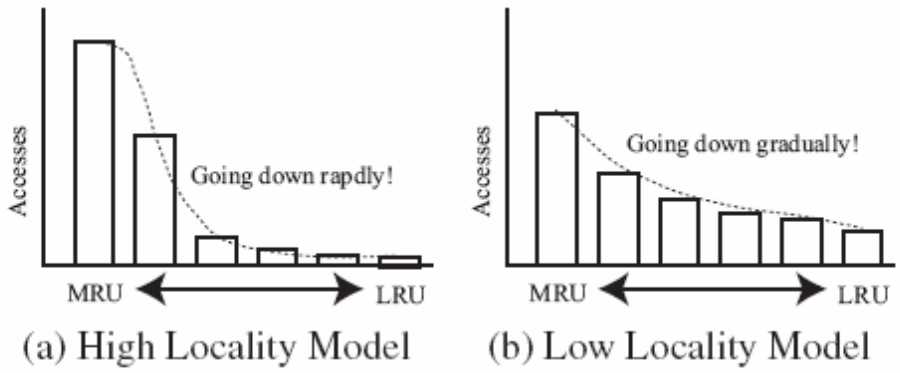


Figura 1.9 – Modelli di località

I grafici riportano sull'ascissa gli stati del campo LRU e sull'ordinata il numero di accessi: quando un'applicazione è caratterizzata da un'alta località la distribuzione degli accessi è concentrata in prossimità dello stato MRU e il numero di accessi decresce rapidamente man mano che si va verso lo stato LRU; quando invece la località è bassa il numero di accessi decresce molto meno rapidamente e il numero di accessi a blocchi LRU non è trascurabile. Quindi il parametro *D* consente di valutare efficacemente il grado di località. Per associare al valore di *D* delle azioni di riconfigurazione, la tecnica way-adapting prevede l'introduzione di due soglie, T_1 e T_2, con $T_1 < T_2$.

Se $T_1 < D < T_2$, dove la metrica *D* rappresenta il grado di località di una cache NUCA , la configurazione corrente della cache è adeguata alla località dell'applicazione e non è necessaria nessuna riconfigurazione; se $D < T_1$ il working set dell'applicazione è contenuto pressoché interamente nella cache e probabilmente c'è la possibilità di ridurre l'associatività senza incorrere in un degrado delle prestazioni: perciò l'azione di riconfigurazione da intraprendere in questo caso è quella di spegnere una via; se $D > T_2$ il working set dell'applicazione richiede una

maggiore dimensione della cache e l'azione da intraprendere è invece quella di accendere una via.

In figura 1.10 è schematizzato il funzionamento del meccanismo di way-adapting in relazione alle soglie T_1 e T_2.

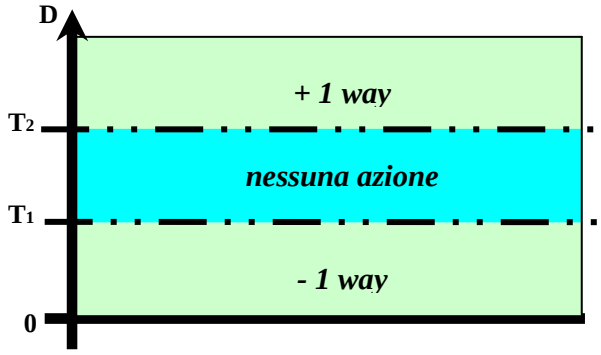


Figura 1.10 – Zone di funzionamento del meccanismo di way-adapting

Per determinare il comportamento globale dell'applicazione in termini di località viene utilizzata una macchina a stati; quest'ultima serve per stabilire la consistenza di una richiesta di riconfigurazione che si presenta ad un certo istante basandosi sui precedenti valori assunti dal parametro *D*. Alla fine di ogni periodo se c'è una richiesta di riconfigurazione viene applicata la macchina a stati e la riconfigurazione viene attuata solo se in accordo col nuovo stato assunto.

Gli stati sono di tre tipi: incremento di una via $(+1)$, decremento di una via (-1) e applicazione della configurazione corrente (0). Se la macchina a stati viene realizzata con n bit per codificare gli stati, sono necessarie almeno $n^2 - 1$ transizioni per passare dallo stato $+1$ allo stato -1.

In figura 1.11 viene riportato il diagramma delle transizioni della macchina a stati, in versione *simmetrica* o *asimmetrica*: la versione asimmetrica è indicata per ottenere maggiori prestazioni poiché il suo

comportamento privilegia le richieste di accensione rispetto a quelle di spegnimento delle vie. Com'è facile intuire quest'ultima versione privilegia le prestazioni a discapito di un aumento del consumo di potenza rispetto all'altra.

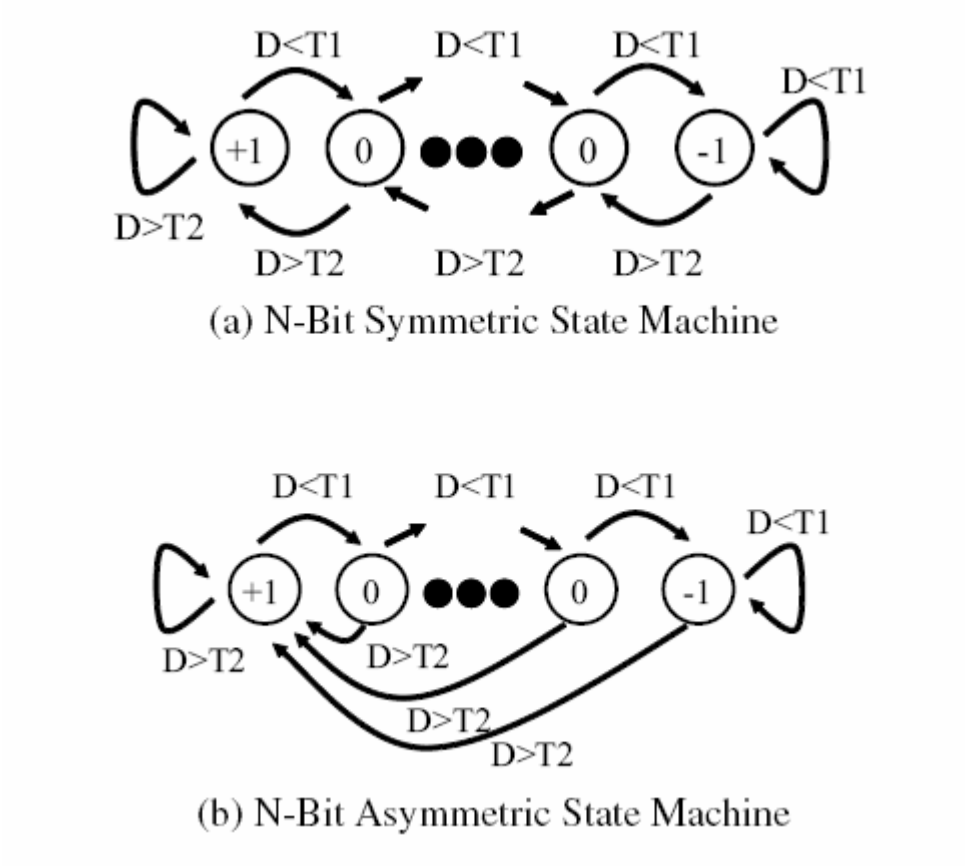


Figura 1.11 – Diagramma delle transizioni della macchina a stati

Nel momento in cui il meccanismo determina lo spegnimento di una via, tale via viene scelta in maniera casuale tra quelle componenti la cache poiché l'analisi che è stata condotta da Kobayashi *et al.* [3] evidenzia come l'utilizzo di politiche alternative per la selezione non determini un aumento significativo delle prestazioni; ovviamente questo meccanismo causa miss aggiuntive quando successivamente vengono riferiti dei blocchi che si trovavano nella via che è stata disattivata.

Nell'applicazione dello way-adapting alle cache D-NUCA questo problema viene attenuato grazie al meccanismo di migrazione dei blocchi.

Il vantaggio della tecnica way-adapting è quello di non dipendere da parametri che sono specifici di una singola applicazione o di una singola tipologia di applicazioni. La stessa flessibilità si ha rispetto alla configurazione della cache a cui si intenda applicare tale tecnica; l'unica restrizione in tal senso è data dal fatto che lo way-adapting è maggiormente efficace quando si considerano cache con alta associatività, poiché è in questa situazione che si può ottenere un maggior risparmio energetico. Gran parte della flessibilità dello way-adapting è dovuta all'adozione della metrica D che riesce a rappresentare efficacemente il grado di località dei programmi senza utilizzare parametri assoluti come, ad esempio, il miss-rate, che variano a seconda dell'applicazione considerata.

In sostanza per implementare lo way-adapting occorre selezionare valori opportuni per le soglie T_1 e T_2 e, relativamente alla macchina a stati, la tipologia (simmetrica o asimmetrica) e il numero di bit per la codifica degli stati. In particolare, per quanto riguarda le soglie, nella maggior parte delle simulazioni che sono state effettuate relativamente a quel lavoro [3] sono stati scelti i valori di 0.005 per T_1 e di 0.0020 per T_2; con queste soglie la tecnica ha mostrato le prestazioni migliori. Per quanto riguarda la macchina a stati, la configurazione con 3 bit per la codifica ha mostrato una maggiore stabilità rispetto a quella a 2 bit, mentre l'impiego di un numero maggiore di bit ha evidenziato un comportamento troppo conservativo. Nel confronto tra versione simmetrica e asimmetrica, infine, si è evidenziata la superiorità dell'asimmetrica in termini di prestazioni; la versione simmetrica, però, è risultata più conservativa in termini di vie attive, offrendo un maggior risparmio energetico.

1.4.1 Definizione di una cache D-NUCA way-adaptable

In questa sezione verrà analizzata in dettaglio l'applicazione della tecnica way-adapting alle memorie cache D-NUCA.
Le memorie cache di tipo D-NUCA, grazie al meccanismo di migrazione dei blocchi, consentono di mantenere lo working set delle applicazioni nei banchi, e quindi nelle vie, più vicini al controller. Da questa considerazione si capisce come questa organizzazione della memoria cache risulti particolarmente adatta all'implementazione dello way-adapting: in effetti, se i dati più frequentemente acceduti si trovano nelle vie vicine al controller, le vie interessate alla riconfigurazione (accensione/spegnimento) sono banalmente quelle più remote. Oltre che per il motivo della semplicità implementativa, le D-NUCA sono adatte allo way-adapting perché offrono la possibilità di ridurre al minimo l'overhead che necessariamente comporta lo spegnere porzioni della cache, cioè l'aumento del numero di miss. Nel caso di cache tradizionali, quando il meccanismo di way-adapting seleziona l'azione di riconfigurazione di spegnimento di una via, tale via è scelta in maniera casuale.
Nelle D-NUCA, invece, la via che viene selezionata è univocamente quella composta dai blocchi a più bassa priorità, cioè con la minore probabilità di essere riferiti. Chiaramente le prestazioni dello way-adapting in questo caso sono legate all'efficienza della strategia di promozione general promotion che, come già anticipato, è solo un'approssimazione della strategia LRU.

Per estendere lo stesso concetto alle cache D-NUCA è necessario definire una metrica per rappresentare lo stato LRU dei blocchi che vengono riferiti, poiché l'ordinamento LRU è solo approssimato.

Infatti nelle D-NUCA bisogna distinguere tra accessi globali e accessi locali (ai singoli banchi): per accesso globale si intende l'accesso all'intera cache, che si ha quando un indirizzo fisico perviene al controller della D-NUCA; poiché un blocco, prima di essere individuato, deve essere ricercato tra i vari banchi del bank set di appartenenza, questo implica, in corrispondenza di ognuno di questi banchi, un accesso locale.

Da un'analisi della distribuzione di hit e di accessi locali per una serie di benchmark [6] si è evidenziato come siano le hit a determinare in maniera efficace l'effettiva località del programma poiché negli accessi entrano in gioco molti fattori legati prevalentemente alle politiche di implementazione delle D-NUCA. La metrica di riferimento dello way-adapting per le cache D-NUCA, quindi, deve essere basata sulle hit. In particolare, il parametro D (introdotto nel paragrafo precedente) può essere esteso a questa tipologia di cache sostituendo MRU_{count} col numero di hit sulla via più vicina al controller, mentre LRU_{count} col numero di hit sulla via più remota (naturalmente la più remota tra quelle attive ad un certo istante).

In formula:

$$\frac{\left(hit\ on\ the\ \ farthest\ way\ _{count}\right)}{\left(hit\ on\ the\ nearest\ way\ _{count}\right)} \tag{1.2}$$

Per uno studio più approfondito si rimanda al lavoro di tesi svolto da Gabrielli [6].

Capitolo 2 – Tuning cache D-NUCA

Questo capitolo chiarirà le strategie seguite durante l'intero lavoro di tesi: il tuning applicato a una cache D-NUCA way-adaptable.
La prima sezione si occuperà delle metodologie di simulazione riguardo l'uso della fase di warmup, prima della fase di run, per le applicazioni prese in esame. La seconda e la terza sezione cureranno in dettaglio gli algoritmi higher local IPC e lower miss rate, descrivendone gli aspetti implementativi. La quarta sezione analizzerà il modello differenziale, scelto come algoritmo di riferimento per il tuning, in quanto dai risultati ottenuti si evince che porta ai maggiori benefici in termini di risparmio energetico.
Infine le ultime sezioni affronteranno i particolari casi di studio, intrapresi nello svolgimento della tesi, e i possibili sviluppi futuri.

2.1 Analisi della fase di warmup

La fase di warmup ha avuto lo scopo di "scaldare" la cache, cioè di minimizzare l'effetto delle *cold misses* all'inizio dell'esecuzione dello workload; in questo modo è possibile osservare il comportamento dell'applicazione in una situazione di regime, poiché all'inizio della fase di RUN la cache è vuota e comporta un numero di miss altrimenti evitabile.
In particolare sono state fatte due prove per ogni benchmark: una prima prova simulando con FFWD e RUN originali, cioè inerenti al file di

configurazione proprio di ogni benchmark[2]; una seconda invece simulando con FFWD uguale al valore originario decurtato del numero di istruzioni scelti per la fase di warmup e RUN uguale al valore originario con in più la fase di warmup scelta.

Con FFWD e RUN si è inteso rispettivamente il numero di istruzioni per cui la simulazione viene eseguita con le funzionalità di timing disattivate e il numero di istruzioni per cui è stata eseguita la simulazione *execution driven*.

A livello pratico l'aggiunta del warmup consisteva nel mettere un *checkpoint* dopo l'ultima istruzione del FFWD. Il numero di istruzioni eseguite fino alla posizione del checkpoint, ci dava la fase di warmup all'interno del RUN.

Nelle prove si è usato il simulatore *sim-alpha way-halting* [6], e le statistiche prese in considerazione sono frutto della differenza tra il valore ottenuto alla fine del RUN e il valore ottenuto alla fine del checkpoint, in quanto dopo quest'ultimo non vengono azzerate le statistiche.

Le figure sotto illustrano quanto detto in precedenza. Con START STATS e END STATS si intende rispettivamente l'inizio e la fine del conteggio delle statistiche (IPC, L2 accesses, miss rate, etc).

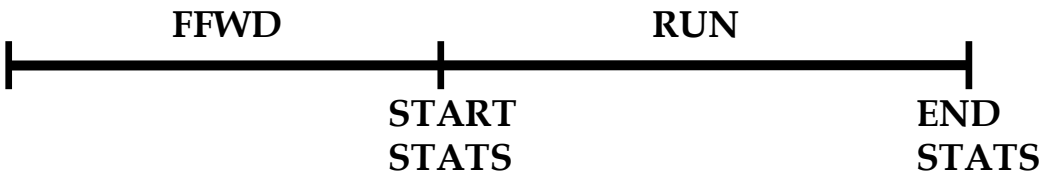


Figura 2.1 – Fase di simulazione senza warmup

[2] Nel capitolo 3 sono descritte in dettaglio le fasi di simulazione per ogni benchmark.

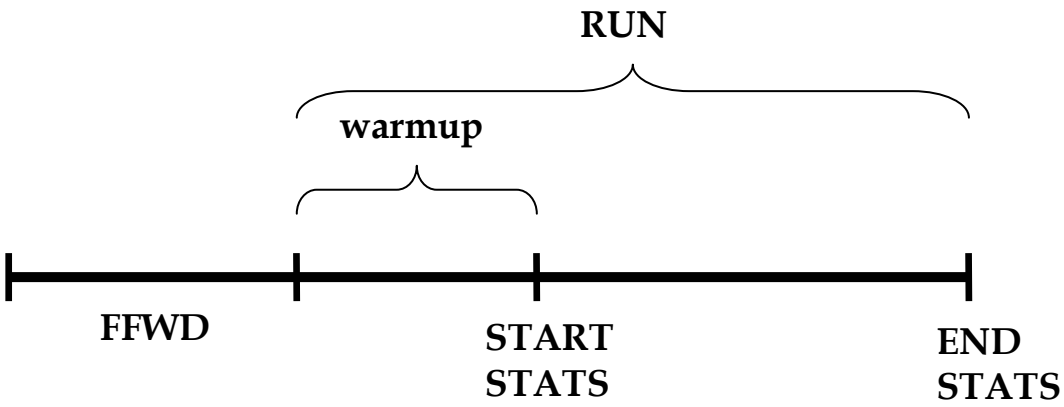


Figura 2.2 – Fase di warmup in relazione alla fase di RUN

I benchmark sottoposti alle due prove sono stati: *gcc*, *bzip2*, *mcf* e *twolf*.
La fase di warmup per tutte le prove è stata assunta pari a 25 milioni di istruzioni.
I risultati hanno dimostrato che per tutti i benchmark in questione si è avuto un miglioramento dell'IPC con l'introduzione della fase del warmup. La tabella 2.1 illustra i risultati ottenuti.

	IPC senza warmup	**IPC con warmup**
bzip2	1.3285	1.4882
mcf	0.3635	0.3938
twolf	0.8441	0.9350
gcc	1.0799	1.1423

Tabella 2.1 – Riassunto IPC nelle due diverse configurazioni

Per completare lo studio sono state fatte delle prove inerenti a diversi valori di warmup; per le tutte le simulazioni il benchmark preso come riferimento è stato *gcc*.

La prova iniziale ha considerato una fase di warmup pari a quella di FFWD e la stessa fase di FFWD pari a zero, cioè si è considerato solo la fase di RUN. La figura 2.3 chiarisce quanto detto.

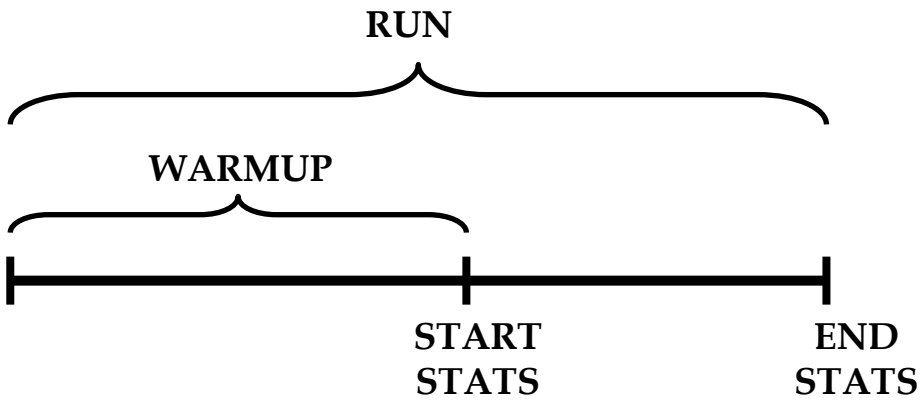


Figura 2.3 – Fase nulla del FFWD

Le prove seguenti hanno riconsiderato la fase del FFWD e per ognuna si è calibrata con un valore diverso la fase di warmup, partendo da valori bassi fino ad arrivare al valore limite di 100 milioni di istruzioni. In dettaglio sono state fatte cinque simulazioni di *gcc* con FFWD uguale a 2357 milioni di istruzioni e warmup:

1. 10 milioni istr.
2. 25 milioni istr.
3. 50 milioni istr.
4. 75 milioni istr.
5. 100 milioni istr.

I risultati ottenuti hanno messo in evidenza che, sia per valori troppo alti che per valori troppo bassi di warmup, le prestazioni in termini di IPC scendono rispetto alle stesse considerando valori di warmup intermedi come possono essere quelli a 50 milioni o 75 milioni. Infatti dalla tabella seguente si vede chiaramente come l'IPC più alto si ha per valori di warmup pari a 75 milioni.

Valori di warmup	IPC (*gcc*)
Wp = FFWD	1.1252
10 Milioni istr.	1.1108
25 Milioni istr.	1.1454
50 Milioni istr.	1.1489
75 Milioni istr.	1.1790
100 Milioni istr.	1.1142

Tabella 2.2 - Riassunto IPC (*gcc*) per diversi valori di warmup

2.2 Algoritmo higher local IPC

Questo e il prossimo paragrafo descrivono i primi passi seguiti nel lavoro di tuning delle soglie di una D-NUCA way-adaptable.

Avendo come riferimento le soglie utilizzate da Kobayashi *et al.*[3], ovvero $T_1 = 0.005$ e $T_2 = 0.02$, si è voluto intraprendere uno studio alternativo riguardo l'utilizzo di nuove soglie da utilizzare successivamente nella fase di way-adapting.

Il nostro studio è inteso come fase di "tuning".

Come accennato nel precedente capitolo (sezione 1.4.1) riguardo l'adozione di una politica way-adaptable, ogni *K* istruzioni (L2 hit in cache, intervallo di *pollrate*) la prima azione che viene effettuata è quella di calcolare il valore della metrica *D*; in base alle soglie T_1 e T_2 si determina poi se al valore della metrica appena calcolata è associata una richiesta di riconfigurazione. Se $D < T_1$ alla D-NUCA viene spenta una via (in gergo si ha una riconfigurazione a -1), se $D > T_2$ viene accesa una via (riconfigurazione a +1) e se *D* è compreso tra le due soglie non viene preso nessun provvedimento (nessuna azione e riconfigurazione a 0).

L'implementazione che sta alla base dell'algoritmo higher local IPC è di scegliere ad ogni passo della fase di tuning, durante l'intervallo di

pollrate, l'azione da intraprendere in base al valore dell'IPC delle tre riconfigurazioni possibili (+1 o -1 o 0). Preventivamente ad ogni intervallo di pollrate si eseguono tre simulazioni, ognuna con la rispettiva azione di +1, -1 e 0; confrontando gli IPC ottenuti dalle statistiche delle tre prove, viene scelta come effettiva configurazione, per quel intervallo di pollrate, l'azione che ha scaturito il massimo IPC. L'IPC calcolato durante le esecuzioni si riferisce all'IPC locale, cioè inerente all'intervallo di pollrate e non all'intera fase di RUN.
La figura 2.4 ne chiarisce i contenuti sopra esposti.

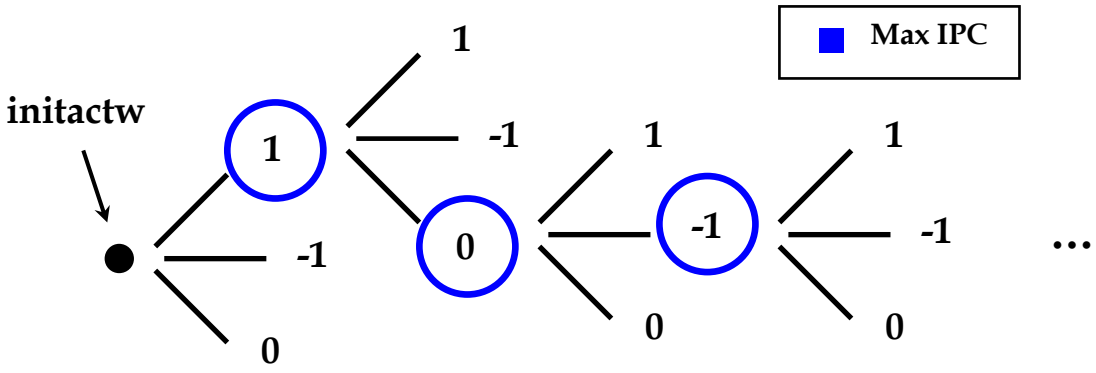


Figura 2.4 – Implementazione dell'higher local IPC (fase di tuning)

In figura le configurazioni cerchiate in blu sono quelle effettivamente scelte durante la fase di tuning; inoltre si è anche specificato il numero di vie attive allo start (per maggiore leggibilità è stato specificato solo al primo step). In base a tale valore, ad ogni checkpoint non è detto che vengano eseguite preventivamente tre simulazioni, dato che se il numero di vie attive è minore di tre le uniche configurazioni possibili sono +1 e 0; analogamente se il numero è maggiore di sette le configurazioni possibili sono -1 e 0.
La figura 2.5 illustra la successione temporale degli eventi di riconfigurazione durante la fase di tuning. L'intervallo che scorre tra checkpoint successivi è il pollrate discusso in precedenza.

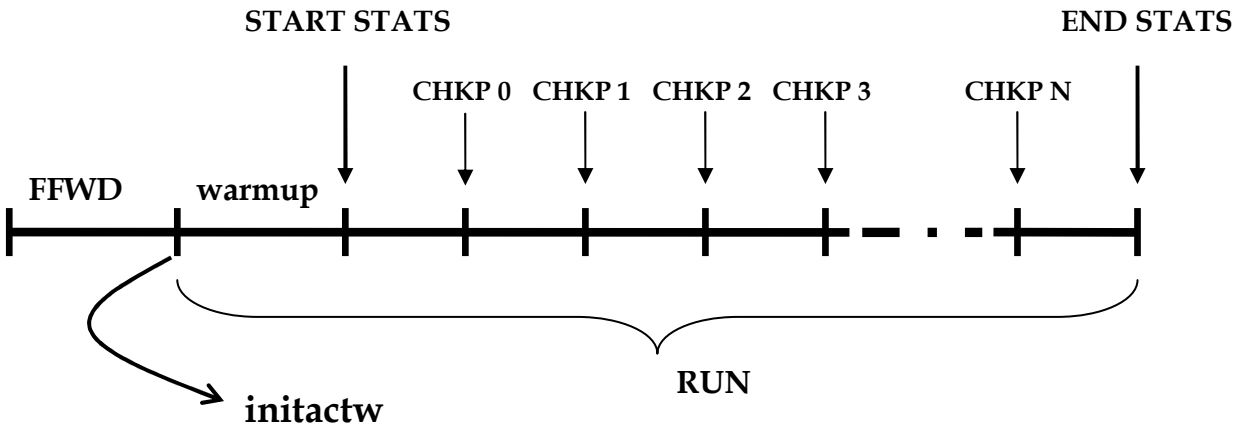


Figura 2.5 – Timing per una cache D-NUCA *way-adaptable*

Con questo algoritmo la richiesta di riconfigurazione non dipende più dalle soglie e dalla metrica *D*, ma viene forzata in base alla scelta dell'IPC ottimo (il maggiore fra gli IPC in questione). Nel qual caso due o più riconfigurazioni hanno lo stesso IPC massimo, viene scelta quella a minore associatività.

In seguito, considerando come incognite le soglie T_1 e T_2 , e ottenute tutte le riconfigurazioni al completamento della fase del RUN, ne viene fuori un sistema di disequazioni così fatto (come esempio sono state prese in considerazione le configurazioni di figura 2.4):

$$\begin{aligned} & T_2 < D_{firstChkpt} \quad T_1 < D_{firstChkpt} \quad (+1) \\ & T_1 < D_{secondChkpt} < T_2 \quad (0) \\ & T_1 > D_{thirdChkpt} \quad T_2 > D_{thirdChkpt} \quad (-1) \\ & T_1 < T_2 \\ & \dots \end{aligned} \tag{2.1}$$

la metrica *D* è nota, poiché viene calcolata alla fine di ogni checkpoint.

Ottenuto il sistema, non rimane altro che risolverlo nelle incognite T_1 e T_2. A proposito di questo ultimo punto, durante lo svolgimento della tesi, sono state affrontate diverse soluzioni. Per non appesantire l'argomento, ne verrà discusso solo uno: le disequazioni pesate.

Risolvere un sistema di disequazioni significa trovare le regioni di spazio che soddisfano l'intero sistema; in questo caso però ci si è trovati di fronte a un sistema irrisolvibile, cioè non con un'unica soluzione. Quindi il problema era di scegliere quale regione adottare per le due incognite T_1 e T_2, andando a vedere le porzioni nel piano con più intersezioni derivanti dalle disequazioni in esame. Infatti, applicando l'algoritmo menzionato sopra, si trovavano soluzioni molteplici differenti tra loro anche di diversi ordini di grandezza. Come accennato prima, la politica adottata è stata quella di applicare un peso a ogni disequazione considerata, in modo tale da rendere più importanti quelle avute con le prime configurazioni scelte. Tutto questo perché le prime configurazioni sono cruciali nell'economia delle prestazioni finali. Sempre con riferimento alla figura 2.4, se (in base alle soglie adottate) decidiamo di scegliere il ramo a +1, nessuno ci dice che abbiamo fatto la scelta giusta, poiché (in base al benchmark utilizzato) può anche essere che il ramo più indicato nella scelta fosse, per esempio, quello a 0. È chiaro quindi che se si sbaglia già alla prima configurazione, poi le prestazioni finali non saranno più quelle auspicate (ne risentirà tutta l'applicazione). Diversamente non si avrebbero particolari fastidi se le configurazioni sbagliate fossero le ultime.

I pesi assunti per le disequazioni sono stati di due tipi: lineari ed esponenziali; le migliori performance si sono ottenute adottando quelli esponenziali, poiché risolvendo il sistema di disequazioni si ottengono soglie per cui, utilizzate nella classica fase di way-adapting, danno risultati vicini a quelli ottenuti con le soglie di riferimento (0.005,0.02).

L'unico inconveniente che si è riscontrato in tale tecnica consisteva nella risoluzione del sistema, in quanto ammettendo diverse soluzioni, si doveva tener conto delle regioni con più disequazioni soddisfatte. Ciò è stato implementato usando un contatore per ogni disequazione, che veniva incrementato ogni volta quella regione si intersecava con

un'altra. Alla fine si sceglievano le regioni con il valore del contatore più alto.

Il problema era dovuto al fatto che le regioni intersecate non erano necessariamente quelle contigue, cioè nel piano vi erano soluzioni differenti in termini di intervalli di valori dell'incognita, per i quali la disequazione era verificata.

In figura 2.6 si riporta la risoluzione grafica per un sottoinsieme di disequazioni (la soluzione è la regione in rosso).

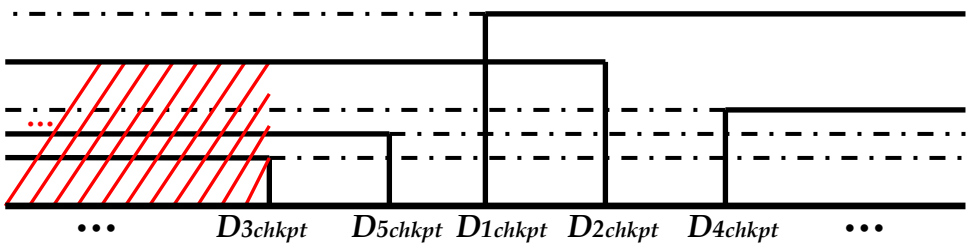


Figura 2.6 – Soluzione grafica di un insieme di disequazioni

Gli studi fatti in precedenza hanno coinvolto i due benchmark *equake* e *gcc* ; la fase di warmup per tutte le simulazioni è stata posta pari a 50 milioni di istruzioni e la fase di pollrate al suo valore di default di 100 mila hit.

In definitiva risolvere questi tipi di sistemi non è stato facile e non ha dato i risultati sperati, poiché utilizzando le soglie trovate[3] nella successiva fase di way-adapting, le prestazioni in termini di IPC non hanno subito miglioramenti, ma anzi in certi benchmark (uno di questi è stato *equake*) si sono evidenziati dei peggioramenti rispetto a quando venivano utilizzate le soglie di default ($T_1 = 0.005$ e $T_2 = 0.02$). La tabella 2.3 sotto riassume i discorsi fatti sinora.

[3] Per *equake* partendo da 2 vie si sono applicate le soglie (0.4,0.5), partendo da 8 le soglie (0.09,0.11).

	2 init. active ways	**8 init. active ways**
(2,0.4,0.5); (8,0.09,0.11)	0.448 IPC	0.463 IPC
(0.005,0.02)	0.460 IPC	0.471 IPC

Tabella 2.3 – Confronto IPC adottando soglie diverse, a parità di vie attive allo start

Per onor di cronaca sono state compiute anche simulazioni senza la fase di warmup e considerate metriche di scelta non basate sul massimo IPC ma su una sua percentuale (limite dell' 1%); inoltre in alcune prove la associatività iniziale è stata posta uguale a quattro ed a otto. Non sono state approfondite perché l'intento era quello di verificare la fattibilità dell'algoritmo in questione.

2.3 Algoritmo lower miss rate

Il principio di funzionamento di questo algoritmo è pressoché identico rispetto al paragrafo precedente, ma se ne differenzia nel modo in cui viene scelta la metrica per decidere ad ogni passo quale configurazione assegnare nel processo di tuning. Precedentemente si è visto che ad ogni intervallo di pollrate veniva scelta la configurazione che portava al massimo IPC nelle statistiche finali dell'applicazione presa in esame; adesso il termine di confronto cambia, prendendo come misura il miss rate più basso.

In verità non si è preso in assoluto il miss rate minore, ma un suo limite che rientrava nella soglia dell'1%.

In formula:

$$Lim = \min Missrate * (100 + threshold)/100 \qquad (2.2)$$

dove il threshold è stato fissato all'1%.

La decisione di prendere il più basso deriva dal fatto che se il programma è soggetto a molte miss, le prestazioni sono ulteriormente penalizzate dal fatto che la risoluzione in miss di un accesso alla cache si determina solo quando sono stati acceduti tutti i banchi del bank set in questione e il messaggio di notifica della miss ha attraversato la rete in senso opposto alla richiesta, giungendo così al controller. Per chiarire questo aspetto, in figura 2.7 si riporta lo schema degli eventi che si succedono nel caso di una miss. Le frecce azzurre indicano il percorso seguito dal pacchetto di richiesta, mentre le frecce rosa indicano il percorso seguito dal pacchetto di notifica di miss. La digitura tag control indica che in corrispondenza di quel banco l'accesso locale è stato interrotto subito dopo l'accesso al campo tag. Quando viene effettuato un accesso ad un banco, infatti, la presenza o meno del blocco è già determinata subito dopo il controllo del tag e, in caso di miss, l'intero accesso può essere abortito anticipatamente, riducendo la latenza complessiva e il consumo di potenza.

Il miss rate calcolato durante le esecuzioni è inteso localmente, cioè inerente all'intervallo di pollrate e non all'intera fase di RUN.

Per il resto valgono gli stessi discorsi fatti sopra.

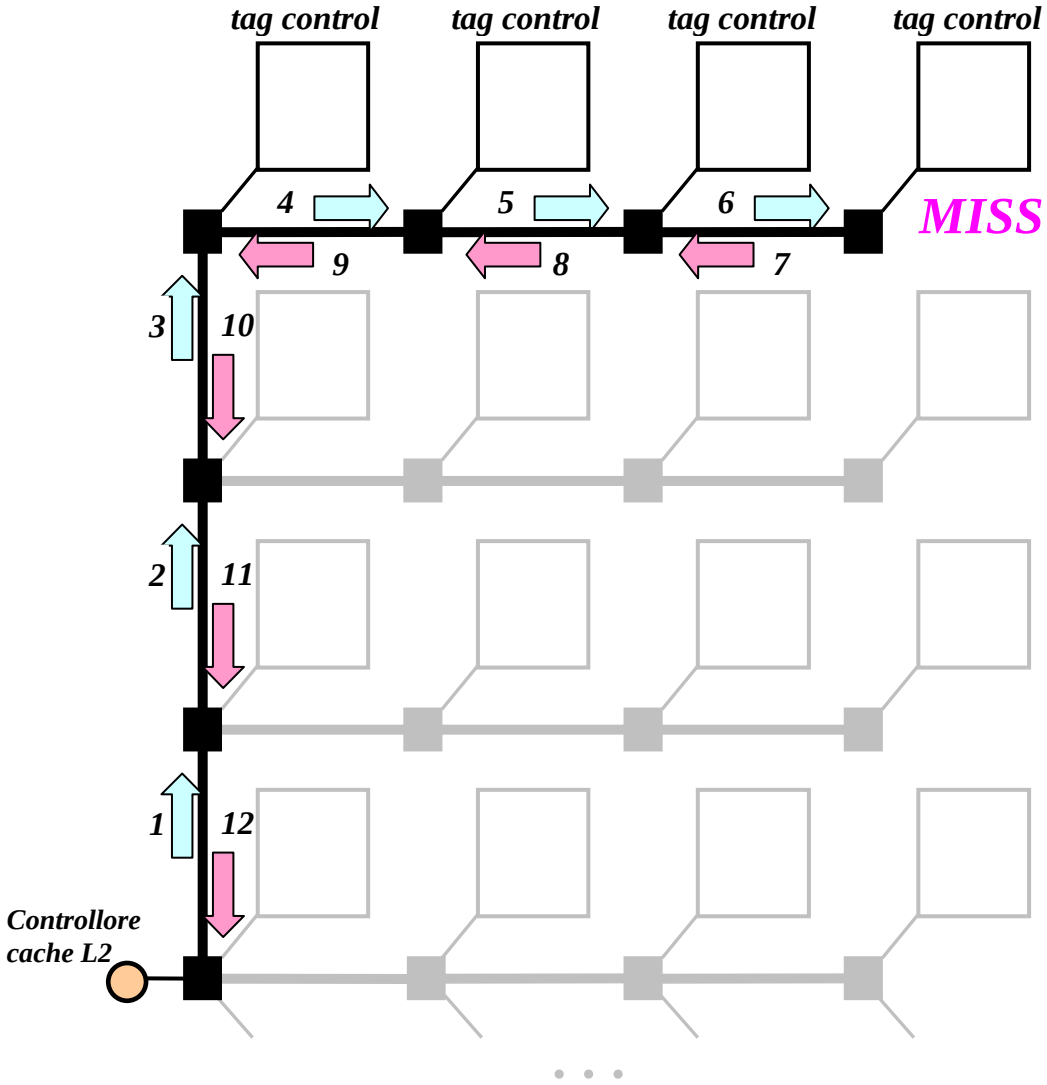


Figura 2.7 – Stati successivi in caso di miss

Anche applicando questo algoritmo i risultati non sono stati quelli sperati, poiché trovate le soglie e sostituite nella fase di way-adapting le performance risultanti non erano tanto migliori rispetto a quelle ottenute con le solite soglie di default. Però c'è da dire che questo algoritmo, rispetto al precedente, nella fase di tuning dava migliori prestazioni in termini di IPC. Per esempio l'IPC, adottando *equake* e partendo da due vie attive, con il primo algoritmo aveva un valore di 0.451, mentre col lower miss rate lo stesso parametro era di 0.455 (+0.88%).

Le opzioni di simulazione sono state le stesse a quelle usate per l'*higher local IPC* e i benchmark utilizzati sono stati *equake*, *cg*, *perlbmk*, *ammp* e *galgel*.

2.4 Modello differenziale in una D-NUCA way-adaptable

Questa sezione descriverà l'ultimo procedimento assunto nella validazione dei risultati. Come si è notato i precedenti paragrafi hanno messo in risalto le difficoltà riscontrate durante la fase di tuning e nella successiva fase di way-adapting, poiché applicando le soglie risultanti le prestazioni non sono state confortanti. Bisognava intraprendere una nuova strada che puntasse più a ottenere risparmi energetici piuttosto che a velocità di esecuzione assolute (in termini di IPC); questo approccio ha portato alla definizione di un algoritmo way-adapting differenziale, che alla resa dei conti ha dimostrato i maggiori benefici, se comparati con le tecniche tradizionali[1],[3].

Per il settore embedded, con l'introduzione sul mercato di dispositivi in grado di supportare molte funzionalità come, ad esempio, gli smartphone o i PDA (Personal Digital Assistant), diventa di fondamentale importanza il consumo di potenza statica, critico per sistemi alimentati a batteria e dall'elevata portabilità.

In una D-NUCA way-adaptable, come menzionato nelle precedenti sezioni, occorre ridurre al minimo l'overhead che necessariamente comporta lo spegnere porzioni della cache, cioè l'aumento del numero di miss; in essa la via che viene selezionata è univocamente quella composta dai blocchi a più bassa priorità, cioè con la minore probabilità di essere riferiti.

Entrando in dettaglio, ci si serve ancora del *lower miss rate*, ma questa volta nell'operazione di tuning vengono apportate delle modifiche sostanziali nella risoluzione dei sistemi di disequazioni. Al posto di seguire la metodologia classica vista prima, e cioè di risolvere direttamente l'insieme nell'equazione 2.1, si procede nel modo seguente: si utilizzano, dalle statistiche finali del *lower miss rate*, le metriche D calcolate in ogni intervallo di pollrate e successivamente vengono ricavate delle nuove metriche differenziali; la nuova metrica *D* è uguale alla differenza tra la metrica *D* di ogni configurazione e la stessa della configurazione successiva; ciò viene ripetuto iterativamente per tutte le configurazioni ammesse dall' applicazione in uso. Il risultato è diviso per la metrica *D* della configurazione successiva.
In formula:

$$\frac{D_{new} - D_{old}}{D_{new}} \tag{2.3}$$

dove *D* è il rapporto tra il numero di hit sull'ultima via (quella più lontana dal controllore L2) e il numero di hit sulla prima via (quella più vicina al controllore L2).
La figura sottostante mostra le azioni intraprese.

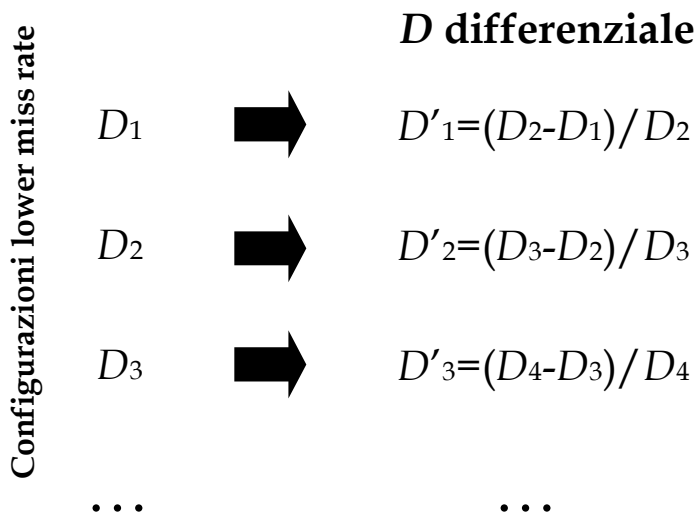


Figura 2.8 – Modifica differenziale della metrica *D*

L'operazione di tuning si completa associando ad ogni nuovo D' la configurazione (+1, -1, 0) opportuna, se maggiore o minore di una determinata soglia in accordo a :

$$\begin{aligned} &(D_{new} - D_{old}) / D_{new} > T_2 \;\; conf = 1 \\ &(D_{new} - D_{old}) / D_{new} < T_1 \;\; conf = -1 \\ &T_1 < (D_{new} - D_{old}) / D_{new} < T_2 \;\; conf = 0 \\ &T_1 < T_2 \end{aligned} \tag{2.4}$$

il risultato adesso è di avere un sistema di disequazioni risolvibile.

I valori di T_1 e T_2 sono le nostre incognite, e sono quelle che ci permettono di decidere se a un dato D' gli associamo la configurazione +1, -1, 0 rispettivamente.

Una volta trovate le soglie adeguate, si verifica il tutto con la tecnica way-adapting opportunamente modificata per testare il modello differenziale (più avanti si esporrà il diagramma di flusso).

Lo studio in esame ha interessato tre diversi benchmark, quali *equake*, *perlbmk* e *ammp* : i primi due verranno analizzati in questa sezione, l'altro in quella successiva. Inoltre le simulazioni portate a termine su di essi sono servite per stilare i risultati finali e avvalorare la fattibilità e la bontà di questo algoritmo.

Le configurazioni di partenza, sia per *equake* che per *perlbmk*, sono state settate in modo da avere una fase di warmup di 50 milioni, l'intervallo di pollrate uguale al suo valore di default (100000 hit), e il numero di vie attive allo start pari a 2.

Di seguito si riportano le tabelle dei risultati in accordo alla definizione delle regioni scelte per le soglie.

Per *equake*:

***D* miss rate**	0,510	**1**	**New *D'***		**1**
2° conf	0,459	**0**		-0,110	**0**
3° conf	0,115	**-1**		-2,987	**-1**
4° conf	0,161	**0**		0,286	**0**
5° conf	0,129	**0**		-0,252	**0**
6° conf	0,732	**1**		0,823	**1**
7° conf	0,154	**-1**		-3,743	**-1**
8° conf	0,224	**0**		0,312	**0**
9° conf	0,143	**0**		-0,571	**0**
10° conf	0,135	**1**		-0,052	**0**
11° conf	0,976	**0**		0,860	**1**
12° conf	0,147	**-1**		-5,638	**-1**
13° conf	0,271	**0**		0,458	**0**
14° conf	0,131	**0**		-1,059	**0**
15° conf	0,177	**0**		0,258	**0**
16° conf	0,857	**0**		0,792	**1**
17° conf	0,308	**0**		-1,781	**-1**

Tabella 2.4 – Lower miss rate e metodo differenziale per *equake*

l'ultima colonna mostra le configurazioni associate al nuovo D' e la terza colonna mostra le configurazioni avute applicando il lower miss rate. Le regioni colorate in arancione mettono in risalto le diverse configurazioni scelte per i due approcci.
Mentre in tabella 2.5 sono inserite le metriche *D'* associate alla conf. 1, 0, -1 rispettivamente.

Ricordando:

- $D' < T_1$, conf = -1
- $D' > T_2$, conf = 1
- $T_1 < D' < T_2$, conf = 0

per T_1 si può scegliere un valore fra *-1.781* e *-1.059* e per T_2 uno fra *0.458* e *0.792*.

										riepilogo	
+1	0,823	- 0,860	0,792							min *D'* 0,792	
0	- 0,110	0,286	- 0,252	0,312	- 0,571	- 0,052	- 1,059	0,258	0,458	max *D'* 0,458	min *D'* - 1,05
-1	- 2,987	- 3,743	- 5,638	- 1,781						max *D'* - 1,781	

Tabella 2.5 – Riepilogo per i valori del *D* differenziale (*equake*)

Per *perlbmk*:

***D* miss rate**	0,046726	**1**	**New *D'***		**1**
2° conf	0,000356	**0**		-130,253	**-1**
3° conf	0,000333	**0**		-0,069	**0**
4° conf	0,000469	**0**		0,289	**0**
5° conf	0,000419	**0**		-0,119	**0**
6° conf	0,000650	**0**		0,355	**1**
7° conf	0,000209	**0**		-2,110	**-1**
8° conf	0,000387	**0**		0,459	**1**
9° conf	0,000492	**0**		0,213	**0**
10° conf	0,000365	**0**		-0,347	**0**
11° conf	0,001059	**0**		0,655	**1**
12° conf	0,000544	**0**		-0,946	**-1**
13° conf	0,001237	**0**		0,560	**1**

Tabella 2.6 - Lower miss rate e metodo differenziale per *perlbmk*

al solito l'ultima colonna mostra le configurazioni associate al nuovo D' e la terza colonna mostra le configurazioni avute applicando il lower miss rate.

							riepilogo	
+1	0,355	0,459	0,655	0,560			min D' 0,355	
0	- 0,069	0,289	- 0,119	0,213	- 0,347		max D' 0,289	min D' - 0,347
-1	- 130,253	- 2,110	- 0,946				max D' - 0,946	

Tabella 2.7 - Riepilogo per i valori del D differenziale (*perlbmk*)

Dalla tabella 2.7 abbiamo invece che per T_1 si può scegliere un valore fra *-0.946* e *-0.347* e per T_2 uno fra *0.289* e *0.355*.

Le soglie che si sono scelte:

$$T_1, T_2 = (-1.2,\ 0.6) \text{ e } (-0.6,\ 0.33). \qquad (2.5)$$

Rispetto alle soglie originali $(0.005,\ 0.02)$, sia la pura associatività media che l'associatività media per il tempo di esecuzione, sviluppati con la tecnica differenziale, risultano migliori di quelli riscontrati dai lavori di Kobayashi. Tutto ciò si traduce in risparmio energetico in grado di compensare le eventuali perdite di performance. Per maggiori delucidazioni si rimanda al capitolo 4.

La scelta delle soglie è da prendere naturalmente con le dovute approssimazioni poiché la procedura utilizzata per selezionare la giusta configurazione per i vari intervalli è puramente empirica.

Per completare il discorso viene descritto il diagramma di flusso seguente, che mostra il modello logico della tecnica way-adapting differenziale.

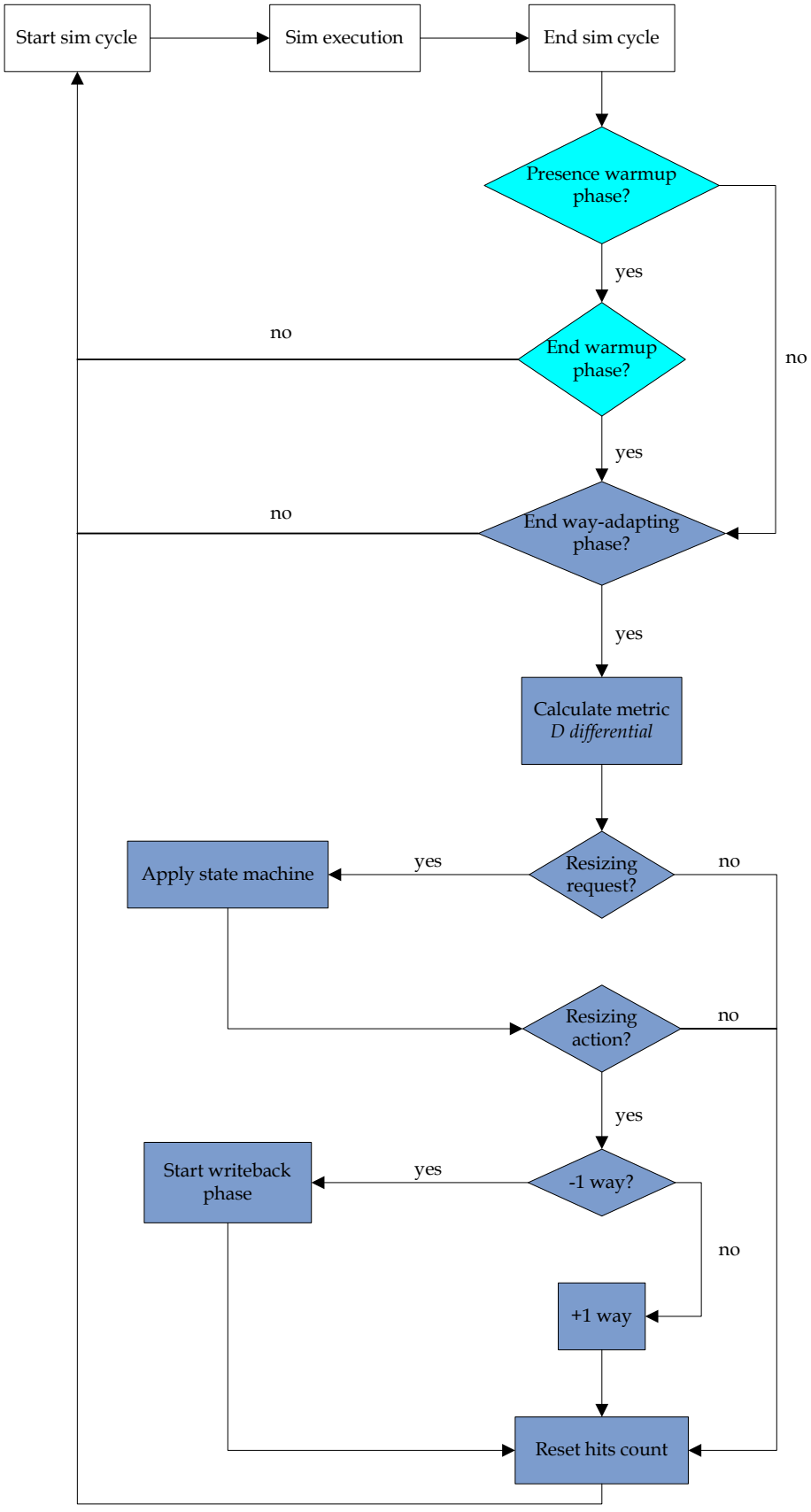


Figura 2.9 – Diagramma di flusso della tecnica way-adapting differenziale

Nel diagramma si sono evidenziati in azzurro chiaro i blocchi inerenti alla fase di warmup e in azzurro scuro quelli inerenti alla tecnica di way-adapting. Dal diagramma bisogna far notare che la macchina a stati riceve in ingresso il tipo di richiesta (accensione o spegnimento di una via) e il vecchio stato, per elaborare il nuovo. Se il nuovo stato corrisponde a quello per cui si ha una riconfigurazione questa viene attuata, con l'accorgimento che, se si tratta di uno spegnimento, l'associatività non cambia immediatamente ma solo dopo la fase di writeback.
Su questa tecnica si ritornerà nel prossimo capitolo, descrivendone i dettagli implementativi circa il simulatore *sim-alpha way-adapting* [6].

2.5 Caso di studio

Il precedente algoritmo, verificate le soglie per l'utilizzo della tecnica way-adapting, è stato provato su tutti i benchmark a disposizione della suite SPEC CPU2000 (enunciati in fondo al capitolo 3). L'average finale adottando le nostre soglie ha messo in risalto come sia l'associatività media che l'associatività media per il tempo di esecuzione (termine di energia) fossero minori rispetto alle stesse adottando le classiche soglie. Ciò non è valso per tutti i benchmark utilizzati, in particolare per *ammp* i parametri sopra detti , applicando la tecnica differenziale, non sono stati affatto minori, ma anzi di gran lunga superiori, rispetto agli stessi utilizzando gli altri benchmark.
ammp rappresenta un particolare caso di studio che necessita di approfondimento.
Questo tipo di situazione è spiegabile perché tale benchmark, finita la simulazione, riporta nelle statistiche finali un tempo di esecuzione in

cicli molto maggiore rispetto alla media; quindi nonostante avesse una associatività media confrontabile con le altre, alla resa dei conti il parametro dell'energia, annoverando pure il tempo di esecuzione, risultava molto più grande delle energie degli altri benchmark.
Quindi la soluzione per ovviare a tale problema è stata quella di studiare singolarmente il suo comportamento. Dato che, come si vede nelle figure 2.10 e 2.11, il *sim cycles* risultava enorme, la soluzione assunta è stata quella di rimpicciolire l'intervallo di simulazione; e per far questo è servita l'analisi preventiva della simulazione di *ammp* con la tecnica way-adapting classica e soglie 0.005 e 0.02.

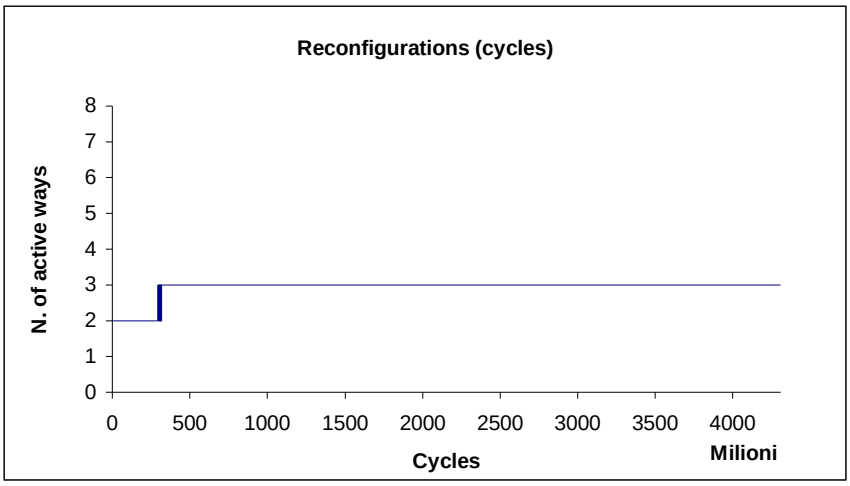


Figura 2.10 – Grafico delle riconfigurazioni per *ammp* adottando $T_1 = -1.2$, $T_2 = 0.6$ (tempo in cicli)

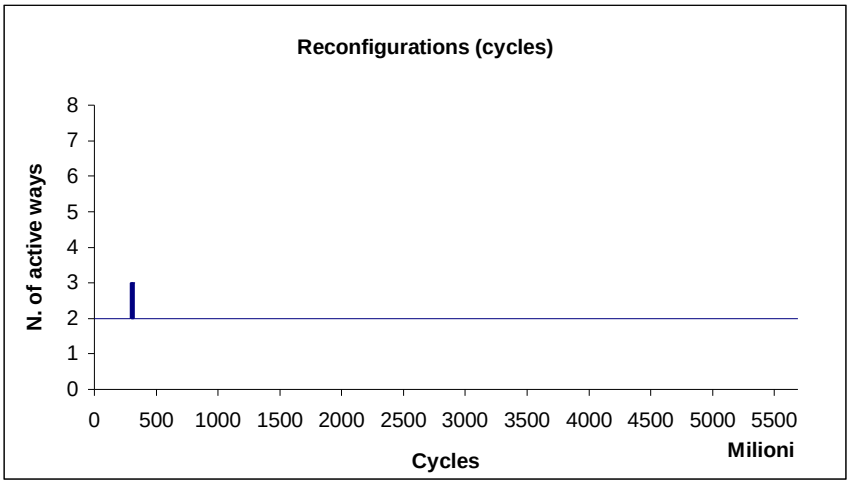
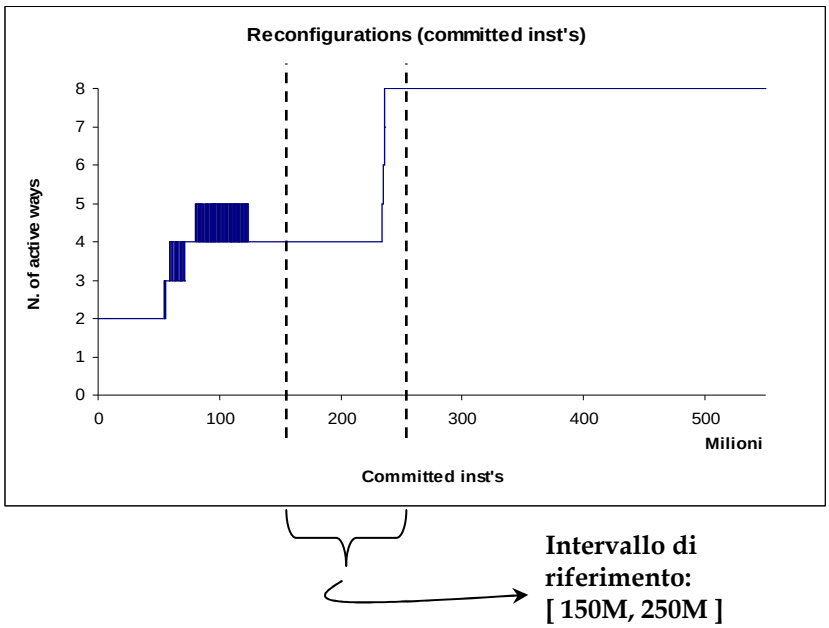


Figura 2.11 - Grafico delle riconfigurazioni per *ammp* adottando T_1 = -0.6, T_2 = 0.33 (tempo in cicli)

Grafico *way-adapting* ammp con T_1 = 0.005, T_2 = 0.02



Dal grafico sopra risulta chiaro come da un certo punto in avanti nella fase di RUN non si hanno più riconfigurazioni, quindi invece di eseguire l'applicazione per intero si è scelto di anticipare la fine della fase di esecuzione. Originariamente *ammp* è stata fatta girare con FFWD pari a 1 miliardo di istruzioni e RUN pari a 550 milioni di istruzioni committed , adesso il suo studio ha riguardato una fase di RUN più piccola (100

milioni) e un FFWD più largo (aggiunti 150 milioni di istruzioni committed); questo ultimo caso per non appesantire la mole di lavoro applicando le tecniche di tuning viste sinora; infatti lo stesso lavoro che si è fatto con *equake* e *perlbmk* è stato fatto anche per *ammp*, perché si pensava che le soglie utilizzate per questa particolare applicazione non fossero adatte, ma come vedremo in seguito le soglie risultanti, applicando la procedura differenziale anche su *ammp* , non hanno differito più di tanto da quelle già utilizzate; dal grafico si capisce come la prima parte della simulazione è caratterizzata da un alto numero di riconfigurazioni, mentre nell'intervallo specificato le stesse sono dell'ordine della decina.

Adottando l'intervallo più corto i risultati sono stati quelli sperati, e rifacendo le simulazioni con le soglie dell'equazione 2.5 il risparmio energetico è valso anche su *ammp*, validando anche su questa applicazione la tecnica differenziale.

Il lavoro di tuning svolto nella sezione 2.4, quindi, è stato effettuato anche su *ammp*, e le soglie che ne sono venute fuori appartengono alla regione [-9.3318 , -0.9853] per T_1 e [0.1219 , 0.4944] per T_2 ; come vedremo più avanti si sono scelti i valori di -1.5 per T_1 e 0.4 per T_2 , molto vicine a quelle già utilizzate (-1.2, 0.6).

Per conformità con tutte le prove precedenti, la fase di warmup, l'intervallo di pollrate e il numero di vie attive allo start non sono state modificate.

Come fatto per *equake* e *perlbmk*, di seguito si riportano le tabelle dei risultati in accordo alla definizione delle regioni scelte per le soglie.

Per *ammp*:

***D* miss rate**	0,8785	**1**	**new *D′***		**1**
2° conf	0,4425	**1**		-0,9853	**0**
3° conf	0,4732	**1**		0,0648	**0**
4° conf	0,0458	**1**		-9,3318	**-1**
5° conf	0,0906	**1**		0,4944	**1**
6° conf	0,2547	**0**		0,6442	**1**
7° conf	0,2867	**0**		0,1116	**0**
8° conf	0,2529	**0**		-0,1336	**0**
9° conf	0,2878	**0**		0,1212	**0**
10° conf	0,2858	**0**		-0,007	**0**
11° conf	0,2531	**0**		-0,1292	**0**
12° conf	0,2882	**0**		0,1217	**0**
13° conf	0,285	**0**		-0,0113	**0**
14° conf	0,2527	**0**		-0,1278	**0**
15° conf	0,2878	**0**		0,1219	**0**
16° conf	0,2848	**0**		-0,0105	**0**
17° conf	0,2528	**0**		-0,1265	**0**
18° conf	0,2879	**0**		0,1219	**0**
19° conf	0,2721	**0**		-0,0580	**0**
20° conf	0,2655	**0**		-0,0248	**0**

Tabella 2.8 - Lower miss rate e metodo differenziale per *ammp*

																riep.	
+1	0,494	0,644														min *D′* 0,4944	
0	-0,985	0,064	0,111	-0,133	0,121	-0,007	-0,129	0,121	-0,011	-0,127	0,121	-0,01	-0,126	0,12	-0,05	max *D′* 0,1219	min *D′* -0,9853
-1	-9,331															max *D′* -9,3318	

Tabella 2.9 - Riepilogo per i valori del *D* differenziale (*ammp*)

La simulazione completa di *ammp* con il lower miss rate ha prodotto ben 204 configurazioni, quindi nella tabella 2.9 sono rappresentate le prime 20 configurazioni per ovvi motivi di spazio.

2.6 Sviluppi futuri

Durante il lavoro di ricerca si sono esplorate un ristretto numero di tecniche per motivi di tempo, poiché l'intento era quello di trovare una metodologia che, se funzionava, risparmiasse energia a dispetto delle tecniche note sinora; nonostante i risultati sono stati confortanti e verificati sotto l'aspetto implementativo, resta però il fatto che possibili studi rimangano ancora deducibili.

Possono essere intraprese sia nuove strade con l'utilizzo di politiche differenti a quella studiata in questa tesi (magari adottando una strategia di tuning incentrata non sul miss rate o sul IPC, ma sul numero di accessi in memoria L2), sia partendo già dal nostro modello differenziale e quindi di proseguire con la ricerca di configurazioni diverse quali possono essere il numero di vie allo start durante le simulazioni o l'intervallo di pollrate o la fase di warmup, visto che qui, per la maggior parte delle prove, sono state prese sempre le stesse.

Capitolo 3 – Strumenti utilizzati

In questo capitolo verrà descritta la metodologia utilizzata per valutare le prestazioni del lavoro di tuning, applicato alla tecnica di riconfigurazione way-adapting in una cache D-NUCA. In particolare si farà riferimento alla piattaforma di simulazione, alle configurazioni del sistema simulato ed infine alle caratteristiche dei benchmark utilizzati.

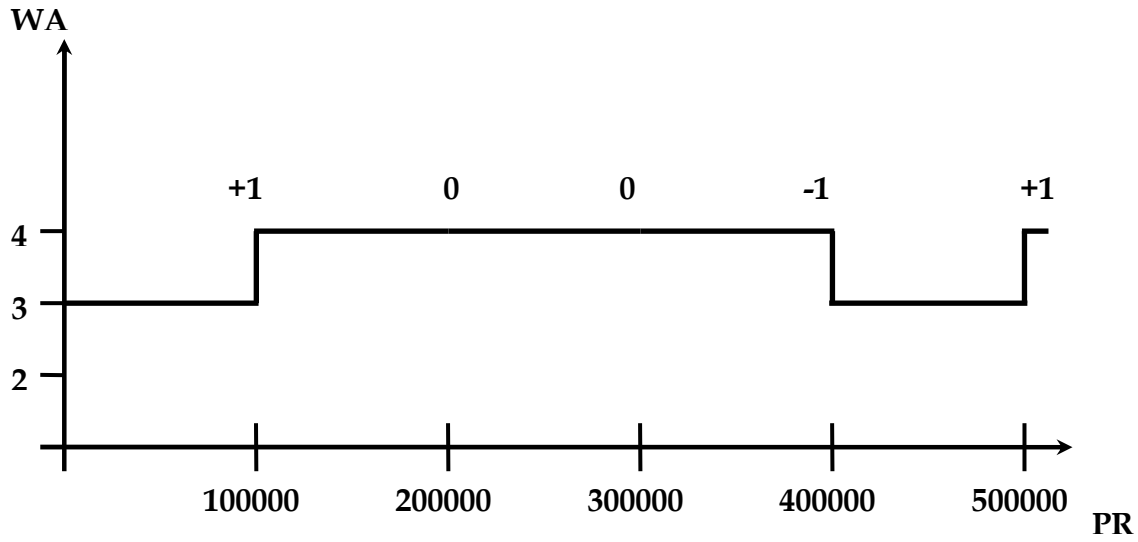
3.1 Simulatori

Per valutare le prestazioni delle varie configurazioni di cache D-NUCA che sono state prese in esame in questo lavoro si è fatto ricorso a diversi simulatori, tutti basati su *sim-alpha* [8]. *sim-alpha* è un simulatore di tipo *execution driven*, in grado cioè di simulare il comportamento dei singoli stadi della *pipeline* di un microprocessore. Il microprocessore su cui è basato è l'Alpha 21264, una cpu a 64 bit superscalare con esecuzione di tipo *out of order* destinata al mercato dei server ad alte prestazioni. I risultati delle simulazioni effettuate su *sim-alpha* sono molto accurati poiché il simulatore è stato validato rispetto ad hardware reale.
Sono tre le diverse versioni di *sim-alpha* che sono state utilizzate: *sim-alpha D-NUCA* per simulare cache D-NUCA tradizionali; *sim-alpha way-halting* per effettuare un'analisi preliminare del comportamento delle applicazioni in presenza o meno della fase di warmup; *sim-alpha tuning* per applicare ad ogni intervallo di pollrate le riconfigurazioni necessarie per il lavoro di tuning; infine *sim-alpha way-adapting* per valutare le prestazioni della tecnica di riconfigurazione way-adapting.
Di seguito descriveremo solo i simulatori sim-alpha tuning e sim-alpha way-adapting, poiché per i nostri scopi sono quelli maggiormente

utilizzati e modificati per applicare il modello differenziale; gli altri due sono stati già abbondantemente descritti nella tesi di Gabrielli [6]. Una sola modifica è stata apportata al simulatore *sim-alpha D-NUCA*, aggiungendo l'opzione per la fase di warmup; questa sarà ben descritta nelle prossime due sezioni.

3.1.1 sim-alpha tuning

Il simulatore *sim-alpha tuning* è stato sviluppato sulla base di *sim-alpha D-NUCA* per implementare la funzione di accensione/spegnimento delle vie della cache; in questo caso gli eventi di riconfigurazione sono specificati in maniera statica, attraverso una particolare opzione del simulatore. In particolare tramite l'opzione –wayadapt:reconf si può specificare il vettore delle riconfigurazioni da applicare alla cache D-NUCA ogni intervallo di pollrate. Ogni evento può essere +1 o -1 o 0.
Se ad esempio (partendo da 3 vie attive) passiamo il vettore [1 0 0 -1 1] e abbiamo settato il pollrate a 100000 hit, gli eventi che ne vengono fuori sono illustrati in questo schema (WA=way active, PR=pollrate):



Di seguito si riportano le opzioni che si possono specificare:

-wayadapt:on <0(off)/1(on)>
-wayadapt:initactw < n. of active ways at start >
-wayadapt:wbsize < writeback buffer size >
-wayadapt:pollrate < n. of NUCA hits per time slot >
-wayadapt:warmup < length of warm-up phase >
-wayadapt:onlyreadhits < count only hits on read (0/1) >
-wayadapt:reconf < reconfiguration sequence >

Con l'opzione -wayadapt:on si attiva il tuning; cioè si permette alla cache D-NUCA di variare dinamicamente il numero di vie attive.
Con -wayadapt:initactw si specifica il numero di vie attive all'inizio della simulazione.
Con -wayadapt:wbsize si specifica la dimensione del buffer impiegato nella fase di writeback. Per dimensione si intende il numero di blocchi da trasferire verso il livello inferiore di memoria.
Con -wayadapt:pollrate si specifica l'intervallo che intercorre tra un evento di riconfigurazione e un altro; l'intervallo è misurato in termini di hit in cache; il valore di default è pari a 100 000 hit.
Con -wayadapt:warmup si setta la fase di warmup, aggiunta alla fase di RUN, per permettere alla cache di "scaldarsi" e quindi di avere più riferimenti in memoria come è stato già descritto nella sezione 2.1.
Con -wayadapt:onlyreadhits si specifica se si vogliono considerare solo le L2 read hit; in questo caso l'opzione si setta a 1.
Riguardo all'opzione –wayadapt:reconf ne abbiamo già discusso all'inizio del paragrafo.

3.1.2 sim-alpha way-adapting

Il simulatore *sim-alpha way-adapting* è stato sviluppato a partire da *sim-alpha D-NUCA* per supportare la tecnica di riconfigurazione way-adapting. Le opzioni che si possono specificare sono le seguenti:

-wayadapt:on <0(off)/1(on)>
-wayadapt:initactw < n. of active ways at start >
-wayadapt:wbsize < writeback buffer size >
-wayadapt:pollrate < n. of NUCA hits per time slot >
-wayadapt:warmup < length of warm-up phase >
-wayadapt:thres1 < low threshold value >
-wayadapt:thres2 < high threshold value >
-wayadapt:smbit < n. of bit for state macine >
-wayadapt:nodeg < 0(off)/1(on) >

Con l'opzione -wayadapt:on si attiva la modalità way-adapting; se non attiva, il simulatore si comporta esattamente come *sim-alpha D-NUCA*.
Con -wayadapt:initactw si specifica il numero di vie attive all'inizio della simulazione.
Con -wayadapt:wbsize si specifica la dimensione del buffer impiegato nella fase di writeback. Per dimensione si intende il numero di blocchi da trasferire verso il livello inferiore di memoria.
Con -wayadapt:pollrate si specifica l'intervallo caratteristico dell'algoritmo way-adapting: è alla fine di ogni intervallo, infatti, che viene aggiornato il valore della metrica *D* e, in caso di richiesta di riconfigurazione, viene applicata la macchina a stati, che poi può schedulare un evento di accensione/spegnimento di una via. Come già anticipato nelle sezioni del capitolo 2, l'intervallo è misurato in termini di hit in cache; il valore di default è pari a 100 000 hit.

Con -wayadapt:warmup si setta la fase di warmup, aggiunta alla fase di RUN, per permettere alla cache di "scaldarsi" e quindi di avere più riferimenti in memoria come è stato già descritto nella sezione 2.1.
Con le opzioni -wayadapt:thres1 e -wayadapt:thres2 è possibile settare le due soglie (T_1 e T_2) che vengono utilizzate per determinare le riconfigurazioni; i valori di default sono pari, rispettivamente, a 0.005 e 0.020.
Chiaramente queste opzioni sono state fondamentali nello spazio di ricerca prodotto per la tesi.
Con l'opzione -wayadapt:smbit si setta il numero di bit della macchina a stati, il cui funzionamento è già stato descritto [6].
Con l'opzione -wayadapt:nodeg si può attivare il meccanismo che evita la degenerazione della tecnica way-adapting in applicazioni con scarsa località e alto miss-rate. Ma questa opzione per lo scopo della tesi non è stata utilizzata.
Infine c'è da annotare la modifica apportata al calcolo della metrica *D*; dovendo applicare il modello differenziale, nel codice si sono aggiunte le seguenti righe:

```
double metric_old = 0.0;
double metric_new = 0.0;
double metric = 0.0;

while(true){
…
metric_old = metric_new;
metric_new = ((double) wa_hits[currActw - 1]) / ((double) wa_hits[0]);
metric = (metric_new - metric_old)/metric_new;
…
}
```

3.2 Configurazione cache D-NUCA

In questa sezione vengono descritte in dettaglio le configurazioni che sono state utilizzate per effettuare le simulazioni relativamente alla cache L2 di tipo D-NUCA e alla tecnica di riconfigurazione way-adapting. Si fa esplicito riferimento alle opzioni dei simulatori che sono stati introdotti in precedenza. Per questioni di spazio se ne tralasciano le descrizioni delle configurazioni del processore e del sottosistema di memoria (cache L1 e DRAM).

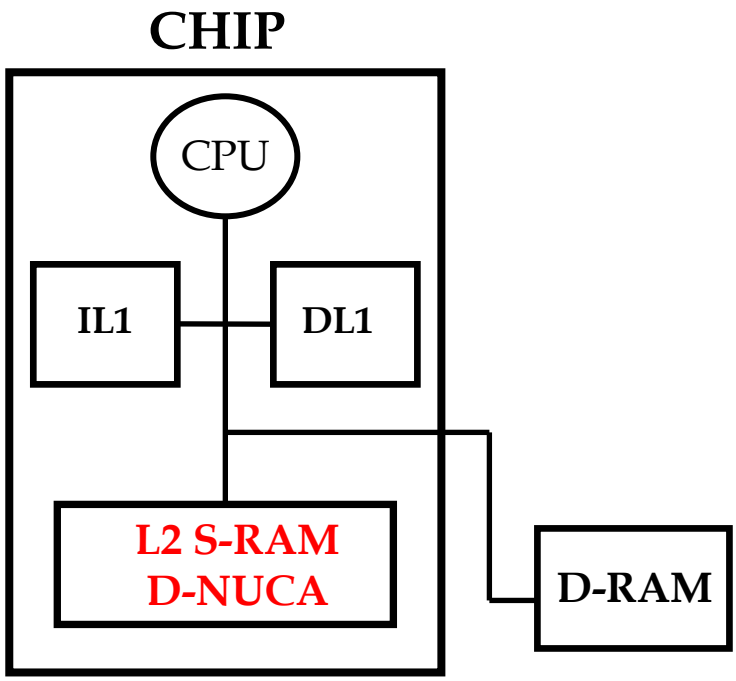


Figura 3.1 – Architettura della memoria a livelli

Per valutare le prestazioni della tecnica way-adapting senza allargare oltremodo lo spazio di ricerca è stata selezionata un'unica configurazione per le cache D-NUCA. Nell'articolo [1] gli autori hanno effettuato le simulazioni su diverse configurazioni di cache D-NUCA,

ognuna delle quali rappresentativa di una particolare tecnologia costruttiva: le 4 configurazioni sono descritte in tabella 3.1.

Technology (nm)	L2 size	Bank org. (rows x sets)	Bank latency (cycles)
130	2MB	4x4	3
100	4MB	8x4	3
70	8MB	16x8	3
50	16MB	16x16	3

Tabella 3.1 – Configurazioni D-NUCA al variare della tecnologia costruttiva

La tecnologia selezionata per questo lavoro è quella con feature size a 70 nm, valore tipico dei processi costruttivi attuali. La cache D-NUCA, come descritto nella tabella 3.1, è di 8 MB, con 128 banchi, organizzati in 16 righe (bank set) e 8 colonne, ed è quindi una 8-way set-associative. La scelta di questo tipo di configurazione è data anche dal fatto che la tecnica way-adapting è particolarmente efficace quando si è in presenza di un'alta associatività.

Vediamo ora quali politiche di implementazione sono state adottate per la D-NUCA.

La politica di *mapping* scelta è la *simple mapping*; un indirizzo fisico che si presenta al controller della D-NUCA può essere logicamente distinto come in figura 3.2:

Figura 3.2 – Indirizzo fisico di una cache D-NUCA

Facendo riferimento alla cache D-NUCA da 8 MB con tecnologia a 70 nm, i singoli campi hanno il seguente significato: la parte *offset* è necessaria per identificare la *word* all'interno del blocco e consta di 6 bit, cioè $\log_2(64)$, essendo i blocchi di 64 byte; la parte meno significativa del campo *index* (*bank set index*) determina il bank set di appartenenza e, in questo caso, consta di 4 bit, $\log_2(16)$, poiché sono 16 i bank set; la parte più significativa del campo *index* determina invece l'indice del set all'interno del singolo banco e consta di 20 bit, $\log_2(20)$,poiché ogni banco è da 128 Kbyte e contiene esattamente 2048 set (questo ultimo valore è dato dal rapporto tra la dimensione dell'intero banco e 512=16x8x4, considerando 4 set *way-associative*); la parte *tag* ha lo stesso ruolo che assume nelle cache tradizionali e occupa i bit più significativi dell'indirizzo.

La politica di *search* selezionata è la *multicast search*.

La politica di *promotion* è la *generational promotion*, con il trasferimento del blocco per il quale si verifica una hit nel banco immediatamente precedente tra quelli componenti il bank set.

Per quanto riguarda la politica di *insertion*, è stata selezionata la *insertion at tail*.

Sull'organizzazione della rete di collegamenti tra gli *switch* e il *controller* della D-NUCA, occorre precisare che la rete non prevede collegamenti verticali tra gli switch se non per quelli corrispondenti alla prima colonna di banchi, come si può vedere dalla figura 3.3.

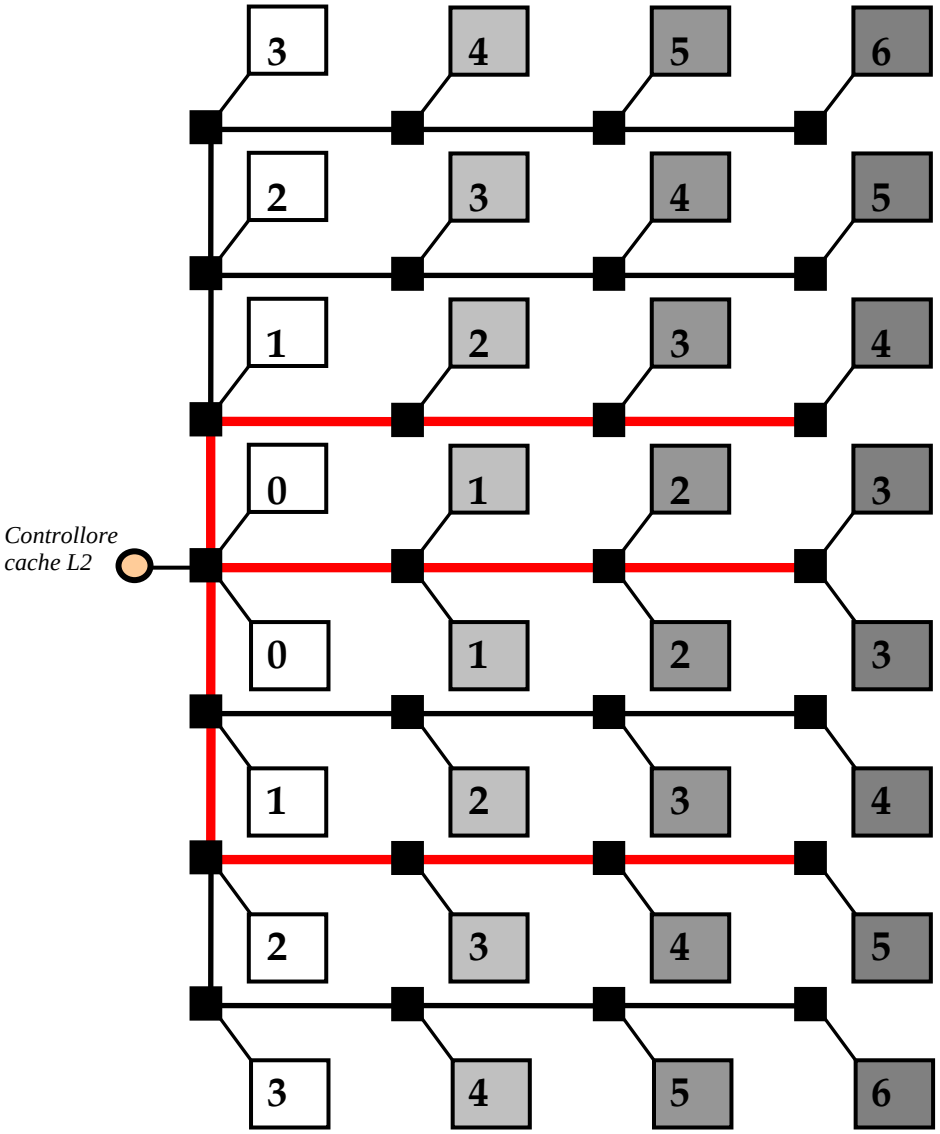


Figura 3.3 – Routing in una cache D-NUCA

Ogni collegamento è bidirezionale nel senso che è composto da due canali completamente separati da 128 bit per trasportare dati e indirizzi. Il traffico nelle due direzioni, quindi, è completamente indipendente per limitare i conflitti.

Il *routing* dei pacchetti prevede che, dato un indirizzo, il pacchetto venga prima inviato sulla riga giusta, attraversando i collegamenti verticali, poi

venga inoltrato ai banchi del relativo bank set attraverso i link orizzontali (i percorsi colorati in rosso). Sempre in figura 3.3 è riportato il ritardo di routing per ogni banco della cache D-NUCA nel caso di assenza di conflitti.

Di seguito si riporta il contenuto del file di configurazione di *sim-alpha* utilizzato per specificare le caratteristiche della cache D-NUCA descritta precedentemente (prima si consiglia di leggere l'articolo di Burger *et al.* [9]):

```
### D-NUCA, 8MB, 16x8, 70nm technology

# L2 cache configuration
-nuca:row 16
-nuca:col 8

-cache:define L2_row0:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row1:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row2:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row3:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row4:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row5:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row6:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row7:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row8:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row9:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row10:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row11:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row12:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row13:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row14:1024:64:0:8:l:3:pipt:0:1:0:Membus
-cache:define L2_row15:1024:64:0:8:l:3:pipt:0:1:0:Membus #8MB
```

```
# L1 cache configuration
-cache:define L1_data:512:64:0:2:F:3:vipt:0:1:0:Onbus
-cache:define L1_inst:512:64:0:2:l:1:vivt:0:1:0:Onbus
-cache:dcache L1_data

-cache:icache L1_inst

# Buses configuration
-bus:define
Onbus:16:1:0:0:16:0:L2_row0:L2_row1:L2_row2:L2_row3:L2_row4:
L2_row5:L2_row6:L2_row7:L2_row8:L2_row9:L2_row10:L2_row11:
L2_row12:L2_row13:L2_row14:L2_row15
-bus:define Membus:16:4:0:0:1:0:SDRAM

# RAM configuration
-mem:define SDRAM
# TLBs configuration
-tlb:define TLB_data:1:32:0:128:l:1:vivt:0:1:0:Onbus
-tlb:define TLB_inst:1:32:0:128:l:1:vivt:0:1:0:Onbus
-tlb:dtlb TLB_data
-tlb:itlb TLB_inst
```

3.3 Benchmark

Tutti i benchmark che sono stati utilizzati per la valutazione della tecnica di riconfigurazione way-adapting appartengono alla suite SPEC CPU2000 e NPB [10]. In particolare la suite CPU2000 ha lo scopo di ricreare uno *workload* significativo per sistemi *general purpose*. I benchmark sono divisi in due categorie: CINT2000 per le applicazioni

con aritmetica intera e CFP2000 per le applicazioni con aritmetica floating-point.

Sono stati selezionati i seguenti benchmark: *bzip2, crafty, eon, gap, gcc, gzip, mcf, parser, perlbmk, twolf, vortex, vpr* (CINT2000); *ammp, applu, apsi, art, equake, facerec, fma3d, galgel, lucas, mesa, mgrid, sixtrack, swim, wupwise* (CFP2000); *bt, cg, sp* (NPB).

CINT2000	FFWD	RUN		CFP2000	FFWD	RUN		NPB	FFWD	RUN
bzip2	744M	1.0B		ammp	1.15B	200M		bt	800M	650M
crafty	1.0B	500M		applu	267M	650M		cg	600M	200M
eon	1.0B	500M		apsi	1.0B	500M		sp	2.5B	200M
gap	1.0B	500M		art	2.2B	200M				
gcc	2.367B	300M		equake	4.459B	200M				
gzip	1.0B	500M		facerec	1.0B	500M				
mcf	5.0B	200M		fma3d	1.0B	500M				
parser	3.709B	200M		galgel	4.0B	200M				
perlbmk	5.0B	200M		lucas	1.0B	500M				
twolf	511M	200M		mesa	570M	200M				
vortex	1.0B	500M		mgrid	550M	1.06B				
vpr	1.0B	500M		sixtrack	1.0B	500M				
				swim	1.0B	500M				
				wupwise	1.0B	500M				

Tabella 3.2 – Intervalli di simulazione dei benchmark

In tabella 3.2 sono riportati gli intervalli di simulazione assunti per ogni applicazione; con FFWD si intende il numero di istruzioni di fast forwarding e con RUN quelle di esecuzione. Sui benchmark in azzurro, anche se sono stati utilizzati per il nostro studio, non se ne conoscono a pieno le particolarità applicative; infatti per tutti si sono usati gli stessi

valori di FFWD e RUN, tranne per *ammp* il cui caso particolare è già stato studiato a parte (sezione 2.5). Bisognava fare una esplorazione simulativa con diverse fasi di simulazione per annotare quali configurazioni andassero bene, ma per motivi di spazio non è stato fatto.
In più durante le simulazioni sono state prese in considerazione per ogni benchmark la propria *eio trace*[4] (external I/O). Queste tracce, generate con il simulatore *sim-eio*, catturano lo stato iniziale dei programmi e tutte le susseguenti interazioni esterne che il programma in esame ha con il sistema operativo.
Quindi per ricreare le stesse modalità di esecuzione, non abbiamo bisogno né di opzioni supplementari, né di file binari, né chiamate di sistema, ma solo delle *eio traces.*

[4] http://www.simplescalar.com

Capitolo 4 - Risultati

In questo capitolo verranno presentati i risultati dell'analisi di tuning, applicato su una cache D-NUCA con tecnica way-adapting differenziale. Come già anticipato nella sezione 3.3, le simulazioni che sono state condotte sono relative ai benchmark della suite SPEC CPU2000 e NPB; anche le modalità di simulazione sono state trattate precedentemente e a questo proposito si veda la sezione 3.1.

Nelle sezioni seguenti verranno mostrati gli esiti delle prove svolte, in termini di IPC (Instruction Per Cycle) e associatività media per tempo di esecuzione al quadrato. I benchmark di cui parliamo sono: *applu, art, bt, bzip2, cg, equake, galgel, gcc, mcf, mesa, mgrid, parser, perlbmk, sp, twolf*; come già detto in sezione 3.3, non si mostrano i risultati su tutti i benchmark utilizzati, poiché per alcuni di loro non se ne conoscono a pieno le particolarità applicative.

Un discorso a parte merita *ammp* già discusso in 2.5 e riproposto in 4.1.

Per ogni benchmark sono state effettuate le seguenti simulazioni, tutte basate sulla medesima configurazione della cache D-NUCA già ampiamente descritta nelle sezioni 2.4 e 3.2 (fase di warmup di 50 milioni e numero di vie attive allo start pari a 2) :

- **wadiff_-1.2_0.6:** simulazione della D-NUCA con meccanismo way-adapting differenziale adottando le soglie $T_1 = -1.2$ e $T_2 = 0.6$;
- **wadiff_-0.6_0.33:** simulazione della D-NUCA con meccanismo way-adapting differenziale adottando le soglie $T_1 = -0.6$ e $T_2 = 0.33$;

- **wa_0.005_0.02:** simulazione della D-NUCA con meccanismo way-adapting normale adottando le soglie di default $T_1 = 0.005$ e $T_2 = 0.02$;
- **DNUCA_BASE_08x08:** simulazione di una D-NUCA base 16x8 a 08MB.

La nomenclatura ha solo lo scopo di far capire quale simulazione è stata eseguita.
Le prestazioni assolute delle diverse tecniche sono riassunte in maniera diretta dal termine IPC. Più è grande, più le performance sono migliori.
Mentre le prestazioni maggiormente interessanti per i nostri scopi, ovvero quelle sul risparmio energetico, sono rappresentate dall'associatività media (della cache D-NUCA) per tempo di esecuzione e associatività media per tempo di esecuzione al quadrato. Più questo valore è minore, più si risparmia potenza.
Inoltre per le esecuzioni citate prima, su ogni benchmark si riportano i grafici relativi agli eventi di riconfigurazione (evento di accensione/spegnimento di vie), riferendo sull'ascissa il numero di cicli di simulazione e sull'ordinata il numero di vie attive, cioè l'associatività della D-NUCA.
La fine del capitolo si concluderà mostrando gli *average*, su tutti i benchmark in questione (anche quelli non usati da Grechi *et al.*), per la metrica IPC, l'associatività media, l'associatività media per il tempo di esecuzione al quadrato e il miss rate. Inoltre si annoteranno, per le metriche sopra menzionate, le differenze percentuali sulle diverse simulazioni fatte, riguardante una stessa applicazione.

4.1 ammp

Nota: nei grafici sono riportati anche i valori inerenti alla simulazione way-adapting differenziale con soglie $T_1 = -1.5$ e $T_2 = 0.4$. Il fatto di utilizzare tali soglie è già stato spiegato nella sezione 2.5; invece le soglie $T_1 = 0.003$ e $T_2 = 0.3$ si riferiscono alla prova fatta senza adottare la tecnica differenziale.

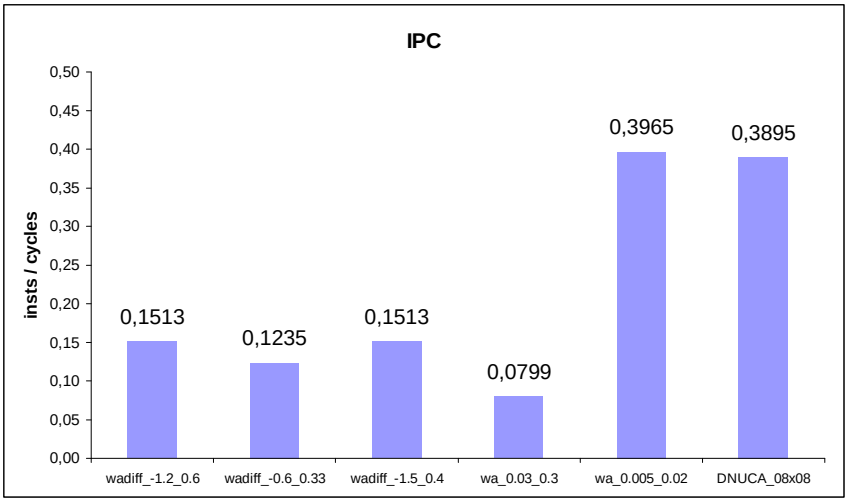


Figura 4.1 – Statistiche IPC relative ad *ammp*

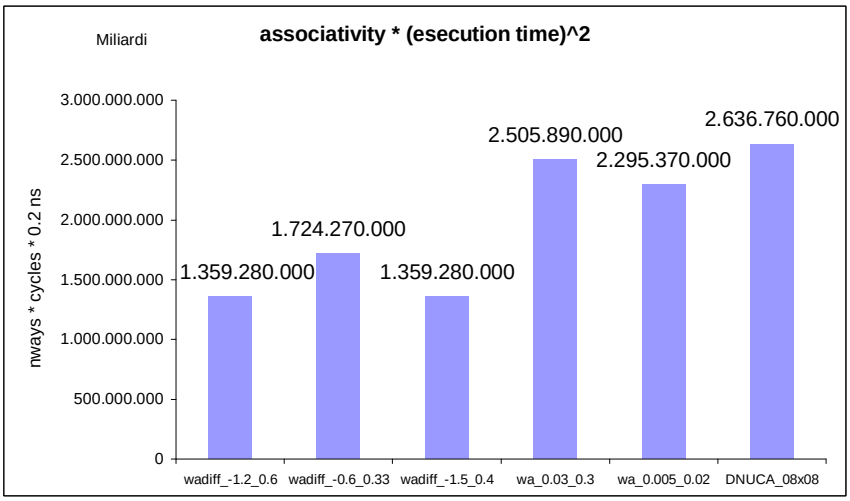


Figura 4.2 – Statistiche $associativity * (\text{esecution time})^2$ relative ad *ammp*

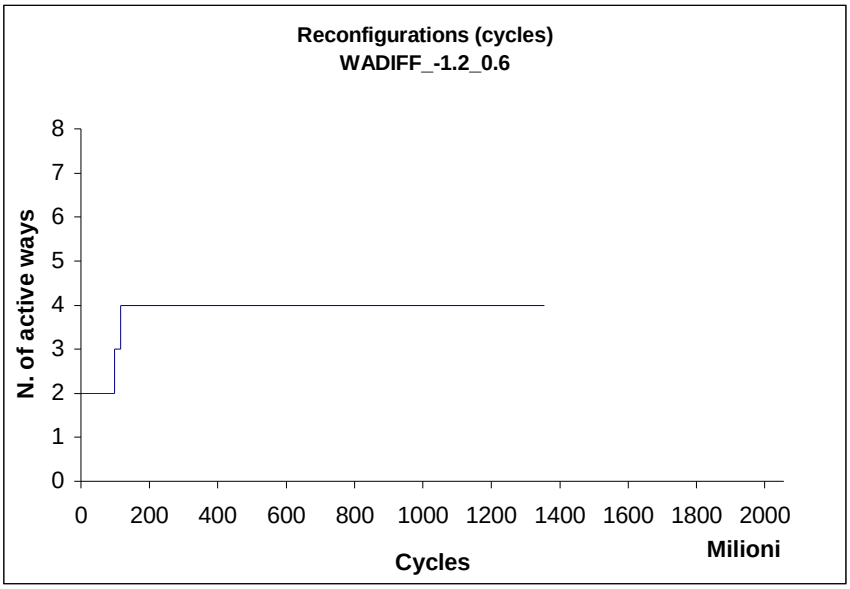


Figura 4.3 – Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative ad *ammp*

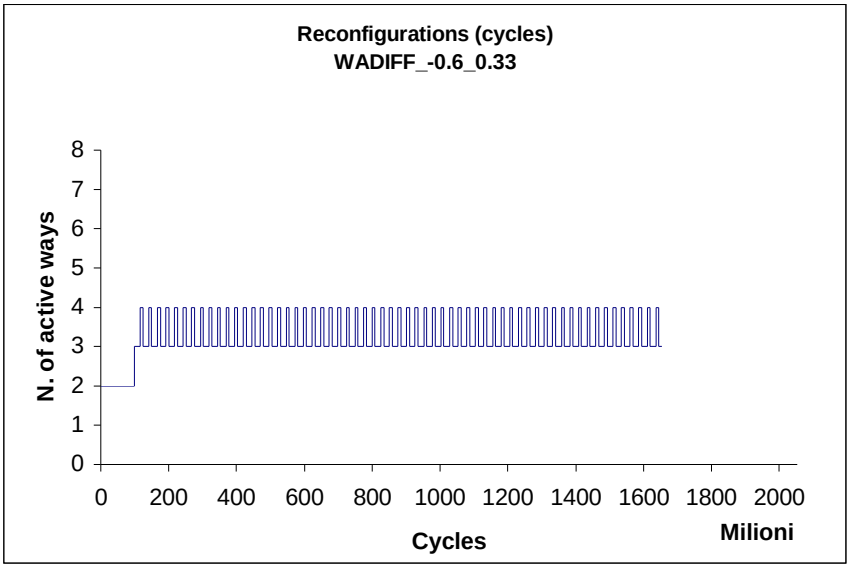


Figura 4.4 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative ad *ammp*

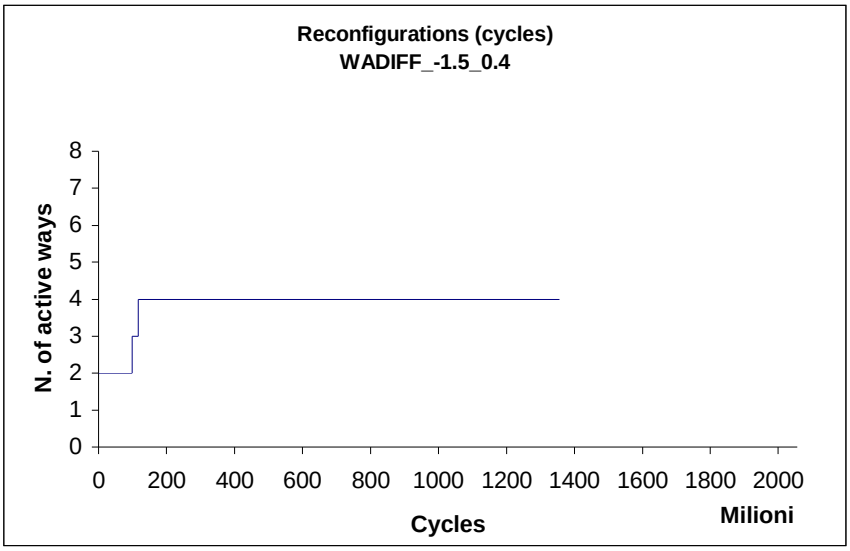


Figura 4.5 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.5$ e $T_2 = 0.4$ relative ad *ammp*

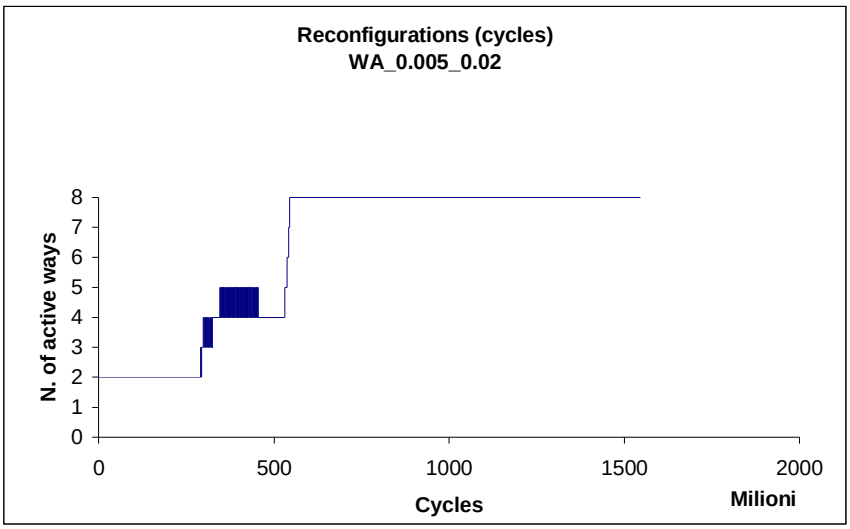


Figura 4.6 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative ad *ammp*

4.2 applu

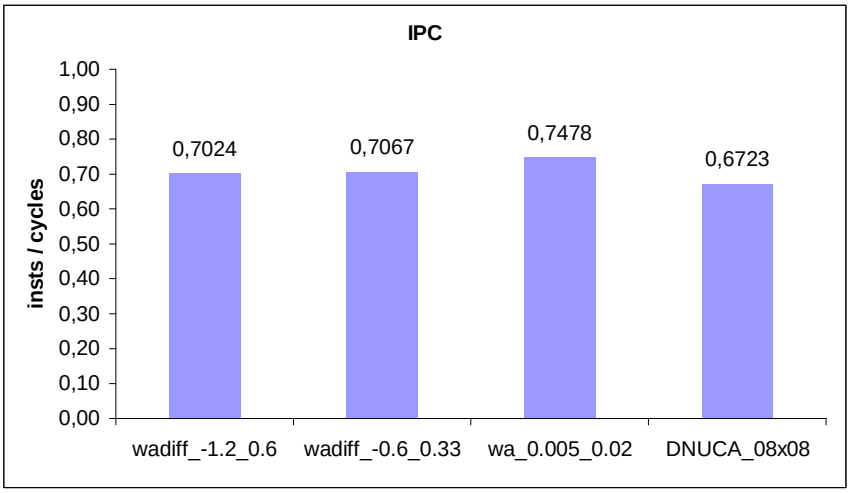


Figura 4.7 - Statistiche IPC relative ad *applu*

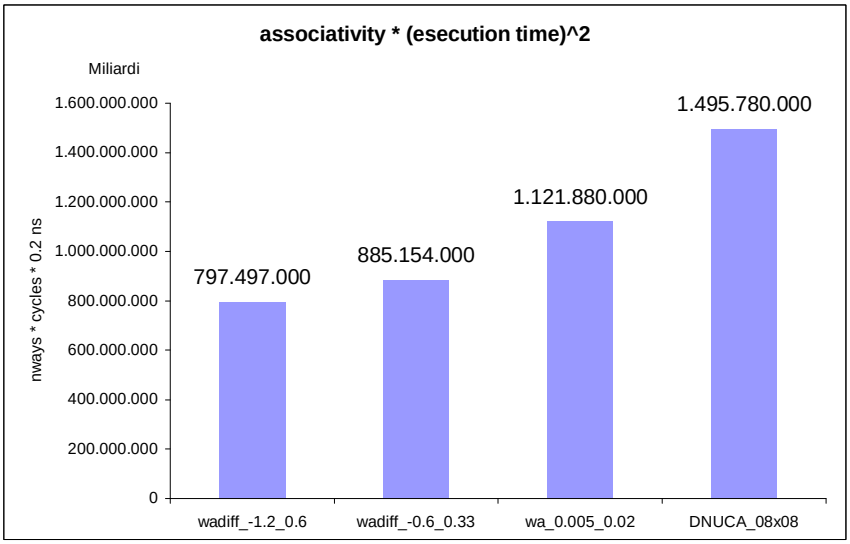


Figura 4.8 - Statistiche $associativity * (\text{esecution time})^2$ relative ad *applu*

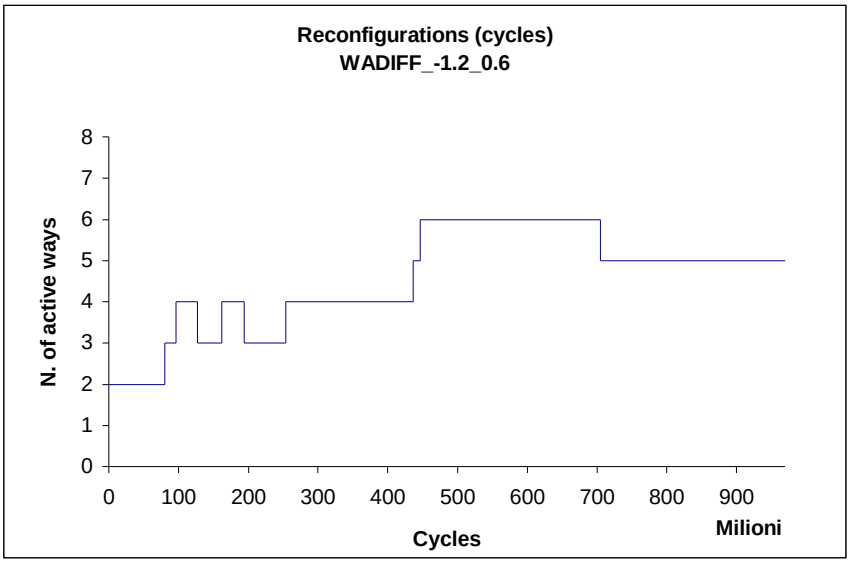


Figura 4.9 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative ad *applu*

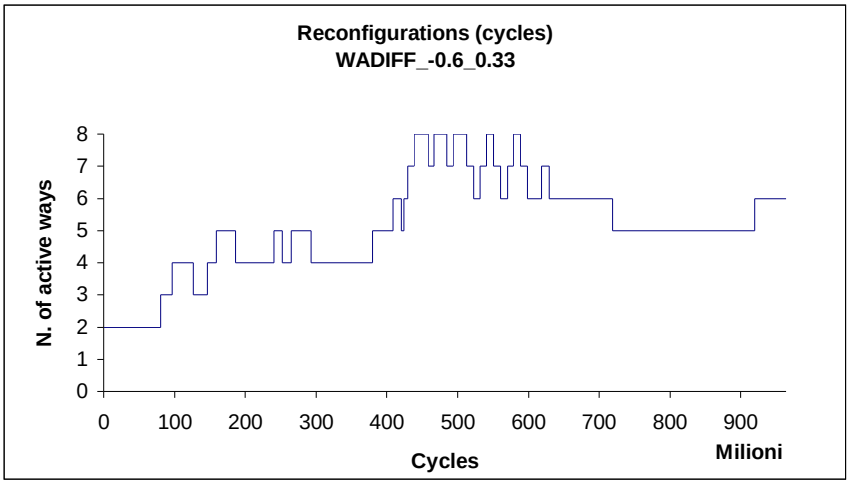


Figura 4.10 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative ad *applu*

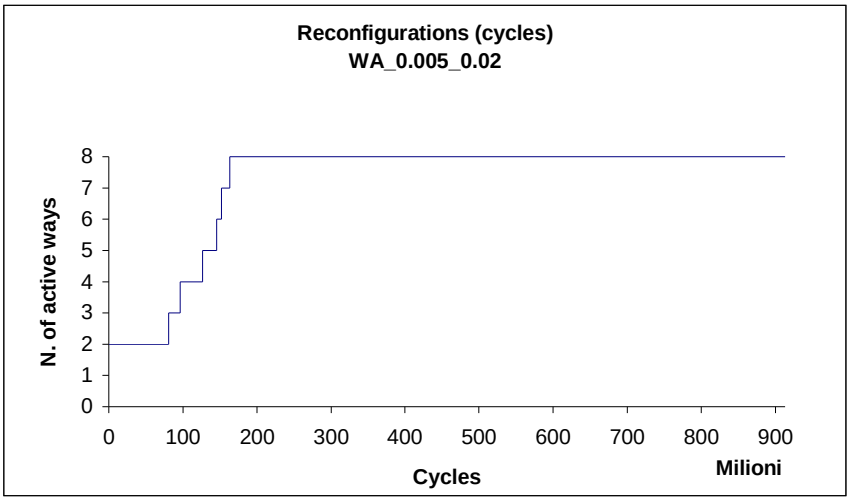


Figura 4.11 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative ad *applu*

4.3 art

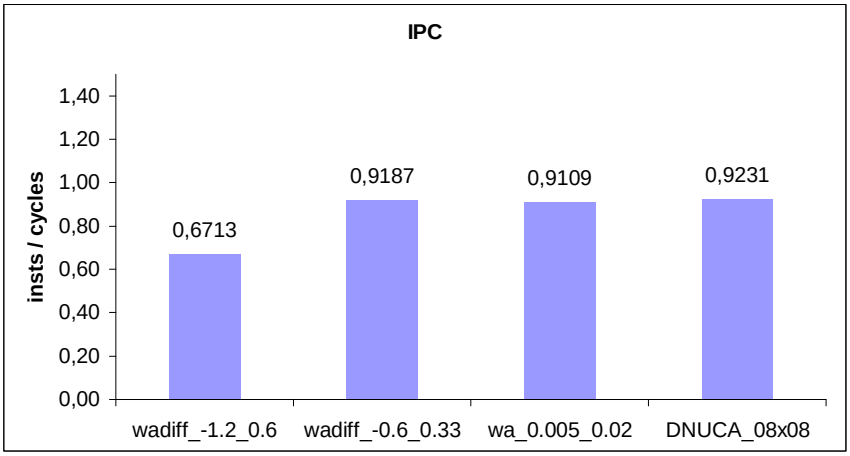


Figura 4.12 - Statistiche IPC relative ad *art*

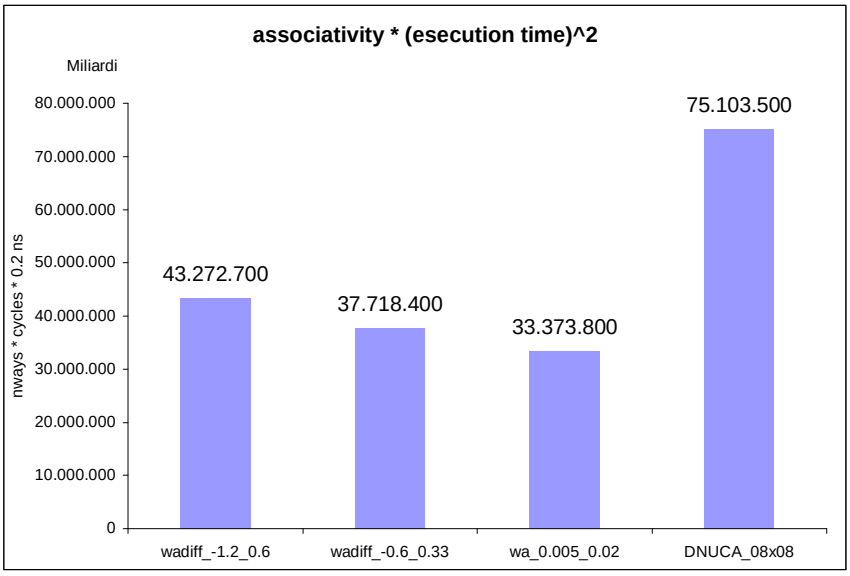


Figura 4.13 - Statistiche *associativity* $*\left(\text{esecution time}\right)^2$ relative ad *art*

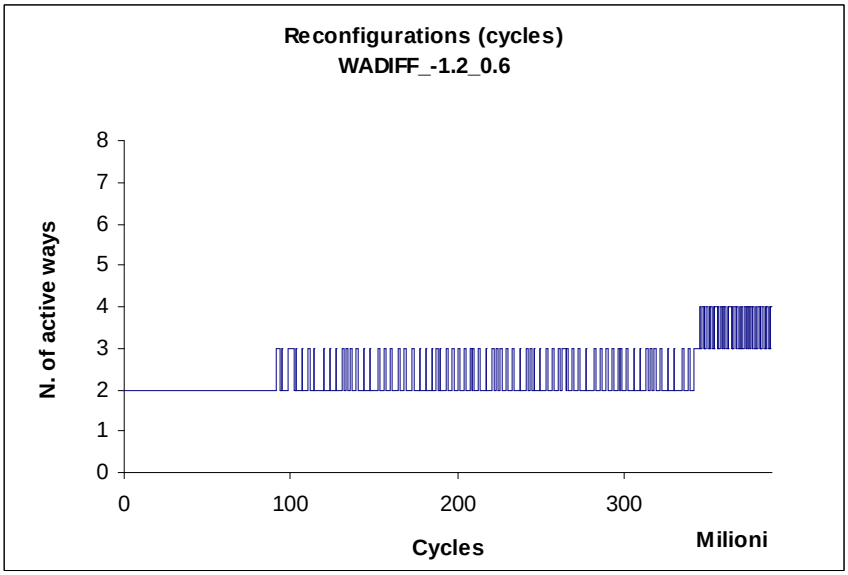


Figura 4.14 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative ad *art*

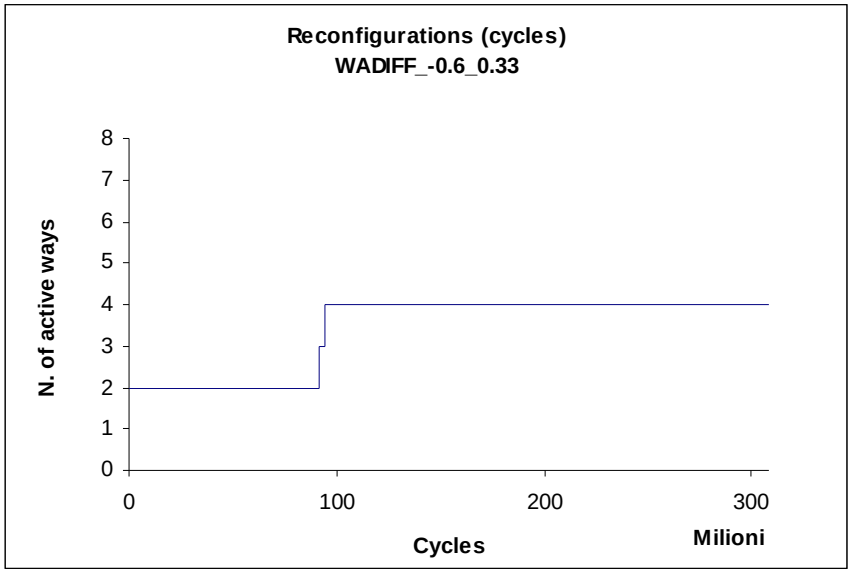


Figura 4.15 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative ad *art*

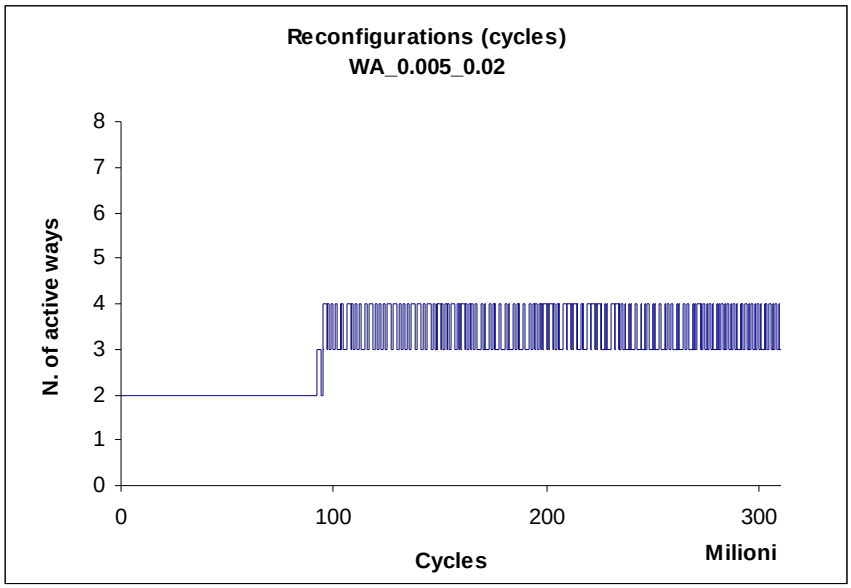


Figura 4.16 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative ad *art*

4.4 bt

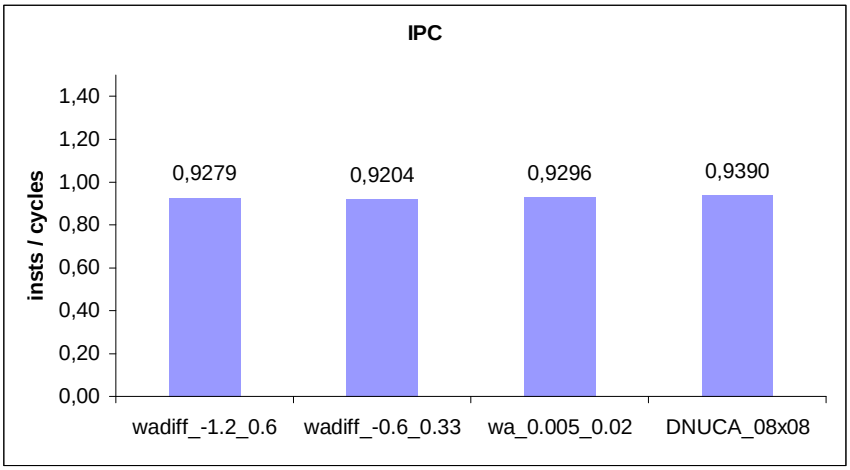


Figura 4.17 - Statistiche IPC relative a *bt*

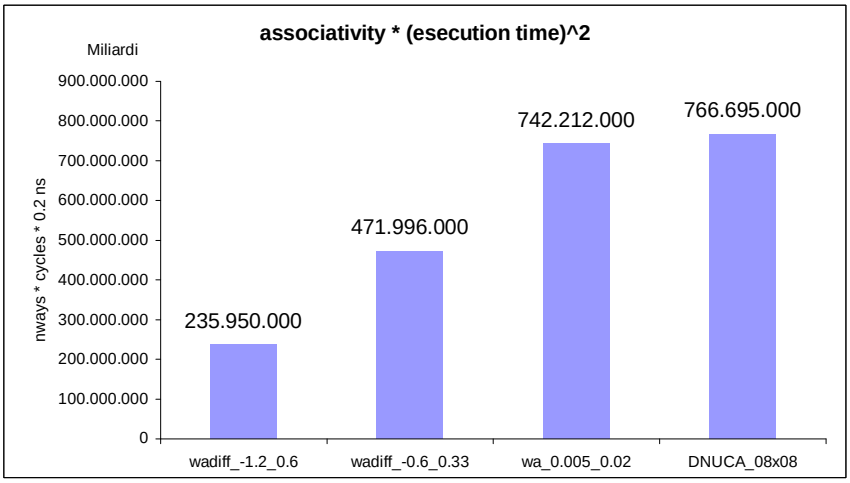


Figura 4.18 - Statistiche $associativity * (\text{esecution time})^2$ relative a *bt*

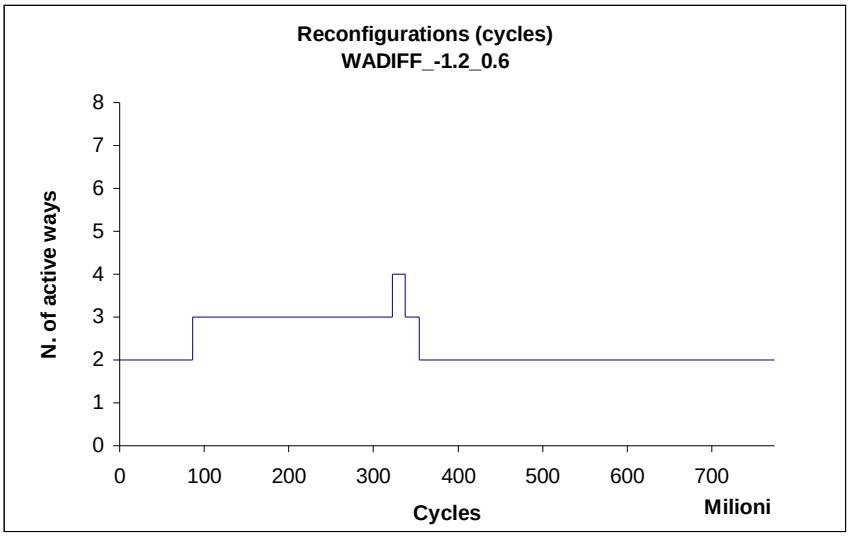


Figura 4.19 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *bt*

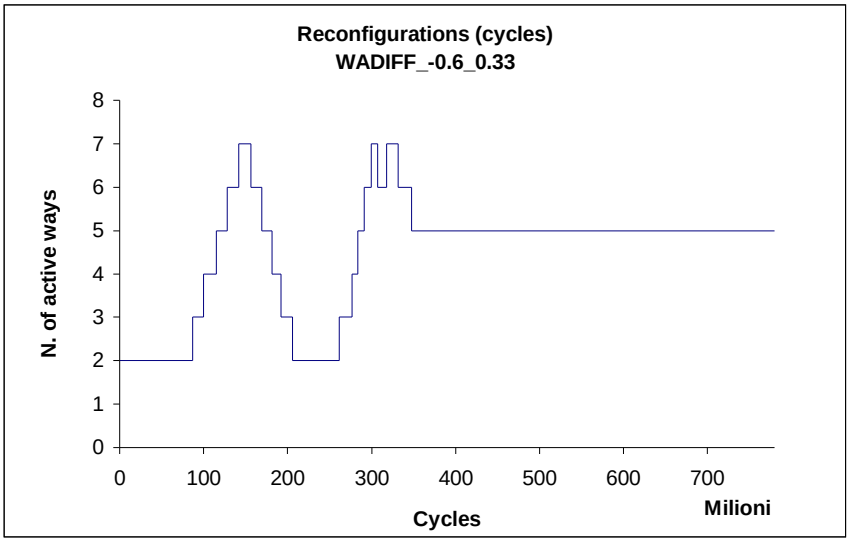


Figura 4.20 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *bt*

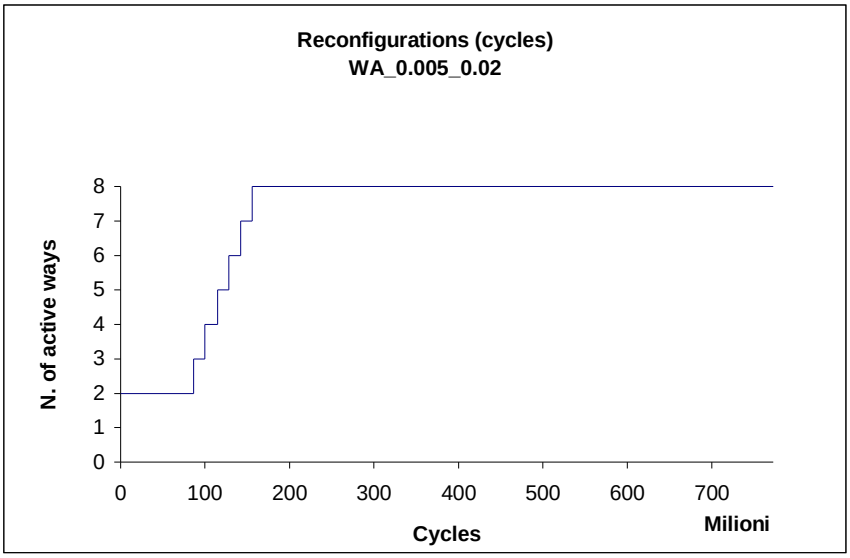


Figura 4.21 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *bt*

4.5 bzip2

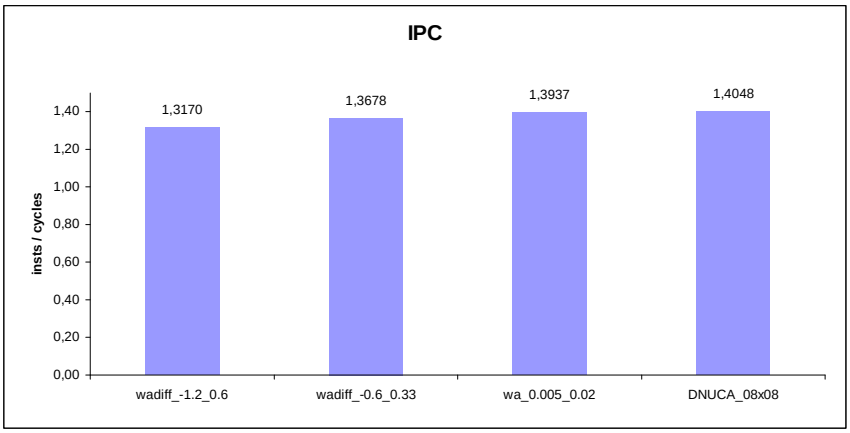


Figura 4.22 - Statistiche IPC relative a *bzip2*

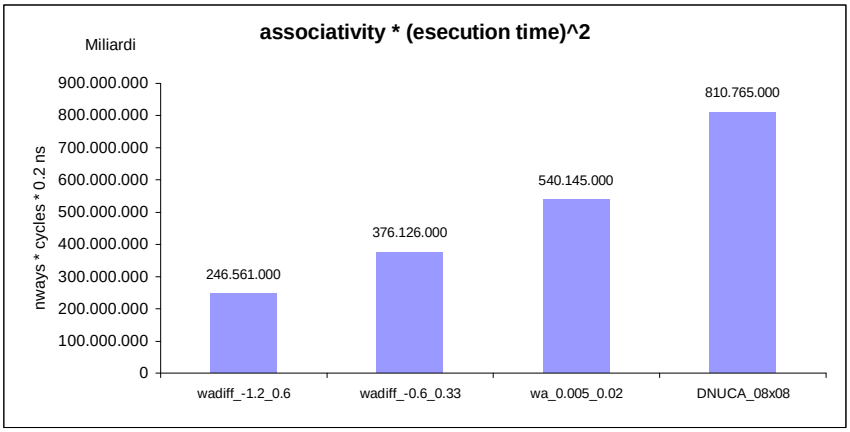


Figura 4.23 - Statistiche $associativity * (\text{esecution time})^2$ relative a *bzip2*

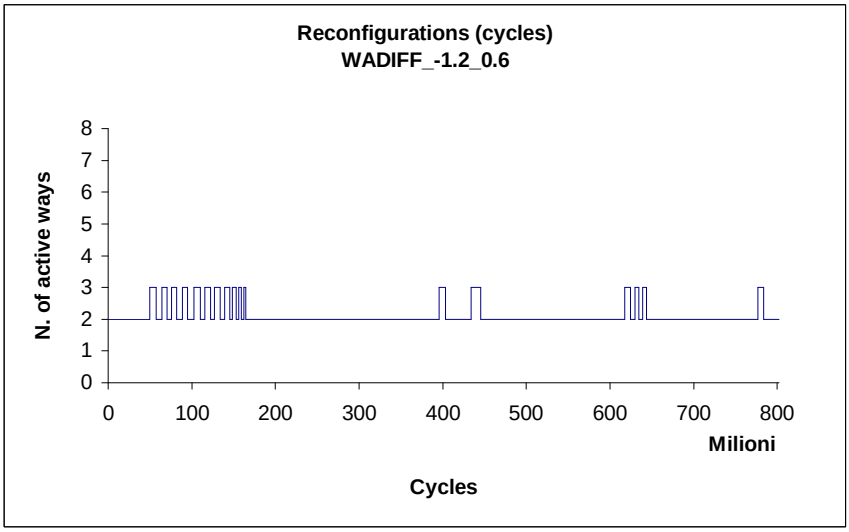


Figura 4.24 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *bzip2*

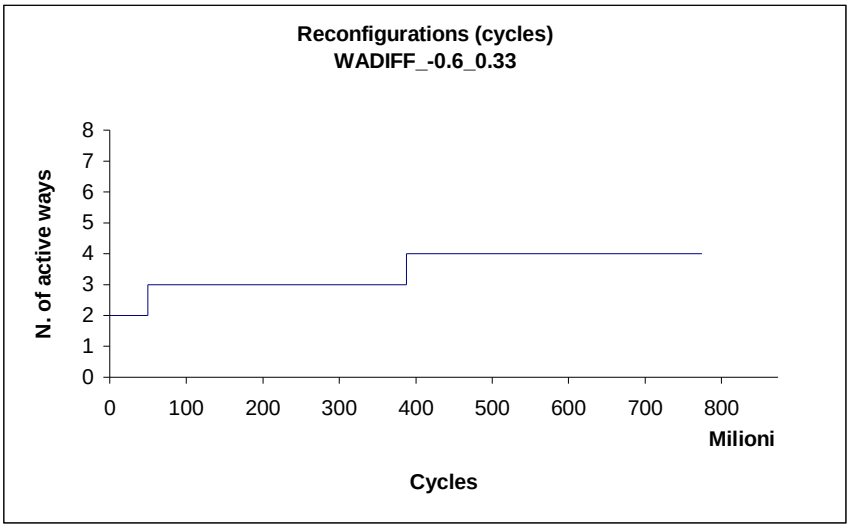


Figura 4.25 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *bzip2*

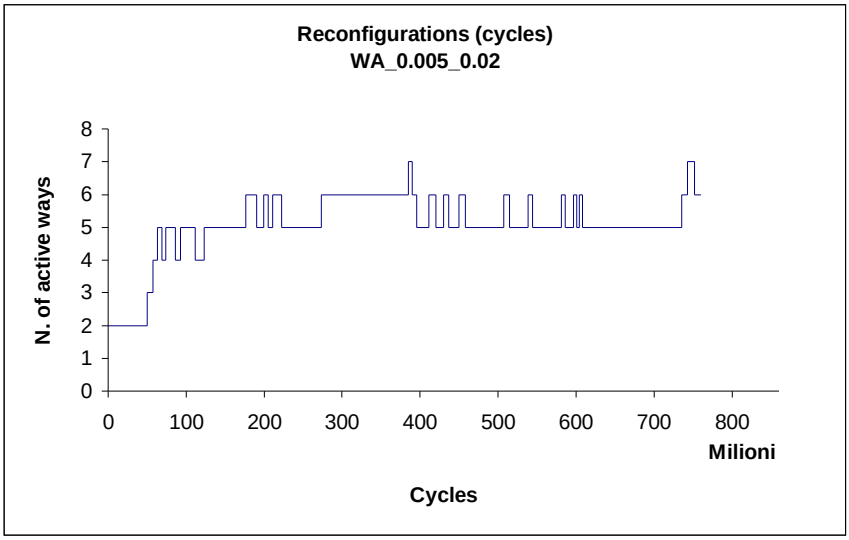


Figura 4.26 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *bzip2*

4.6 cg

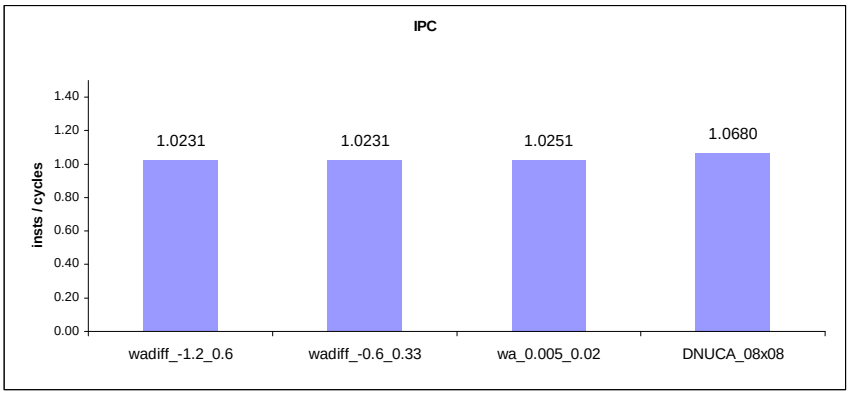


Figura 4.27 - Statistiche IPC relative a *cg*

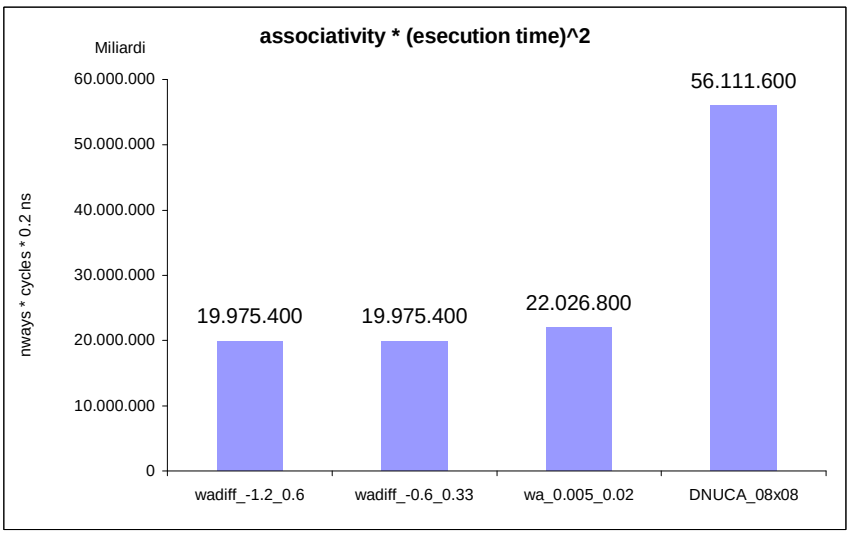


Figura 4.28 - Statistiche $associativity * (\text{esecution time})^2$ relative a *cg*

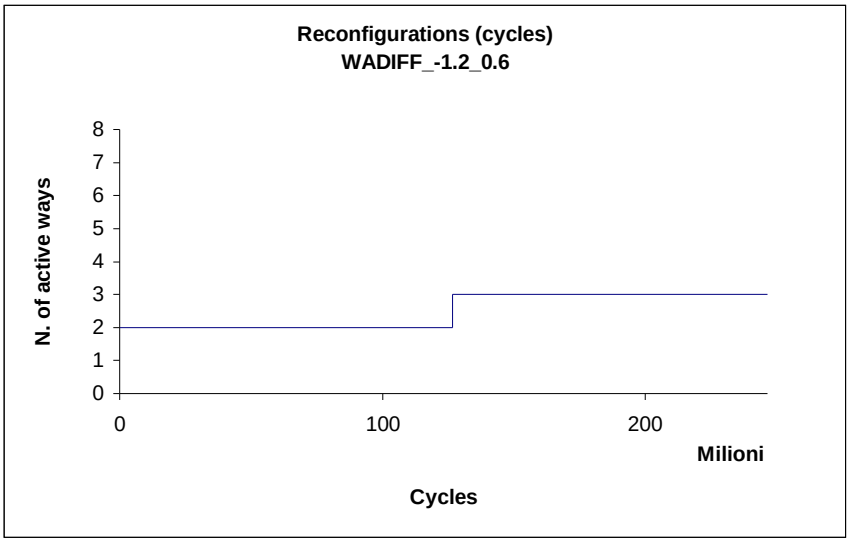


Figura 4.29 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *cg*

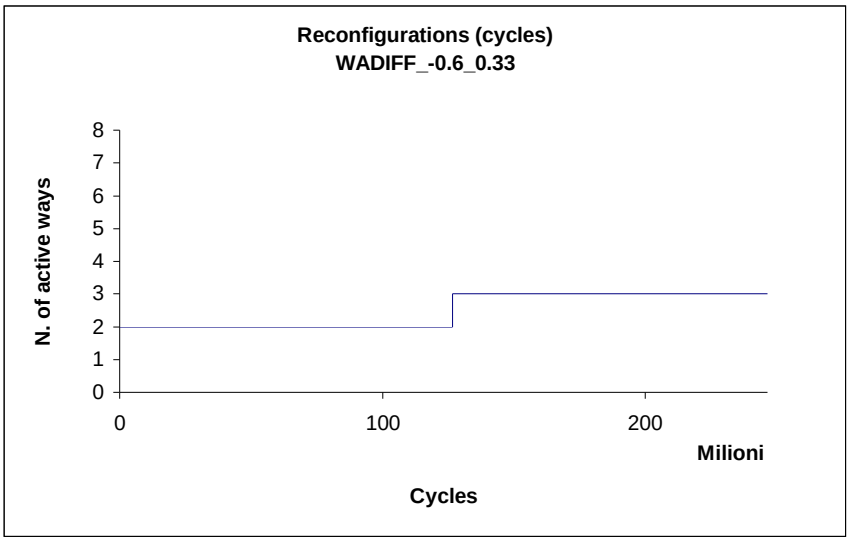


Figura 4.30 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *cg*

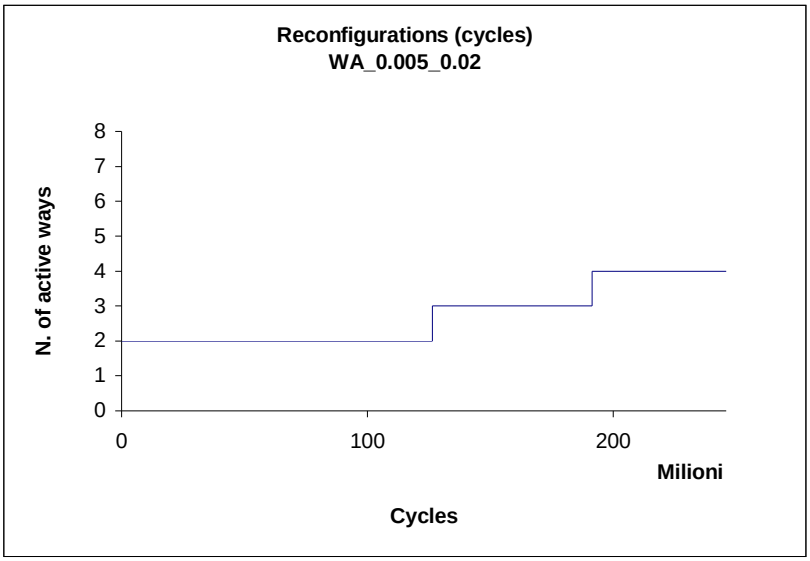


Figura 4.31 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *cg*

4.7 equake

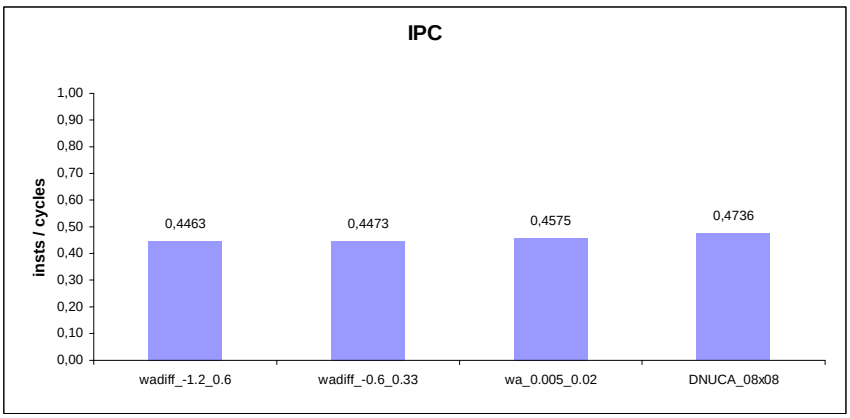


Figura 4.32 - Statistiche IPC relative ad *equake*

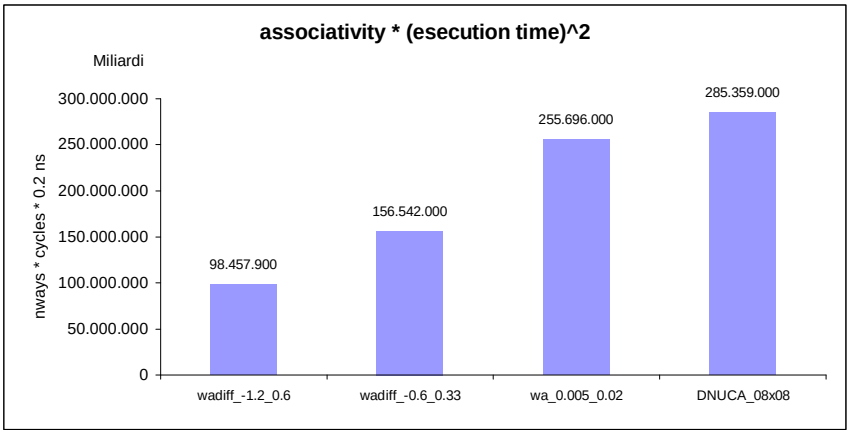


Figura 4.33 - Statistiche $associativity * \left(\text{esecution time}\right)^2$ relative ad *equake*

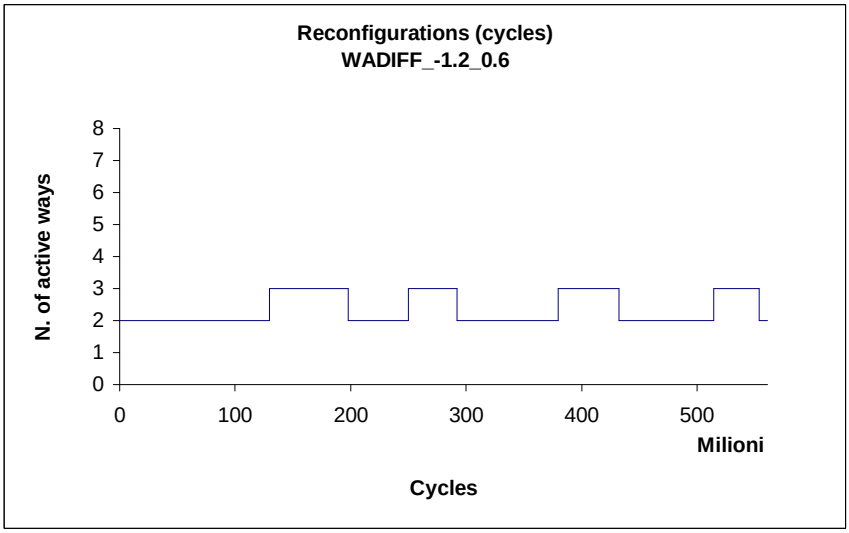


Figura 4.34 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative ad *equake*

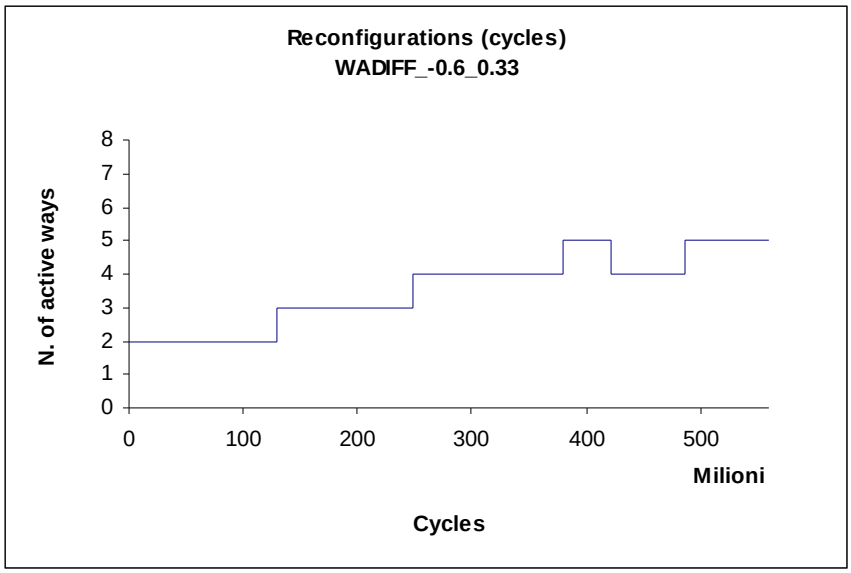


Figura 4.35 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative ad *equake*

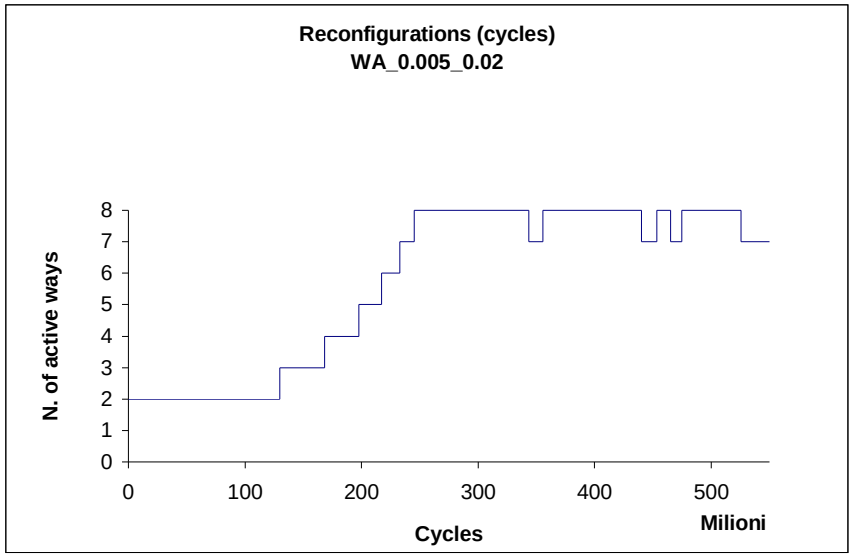


Figura 4.36 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative ad *equake*

4.8 galgel

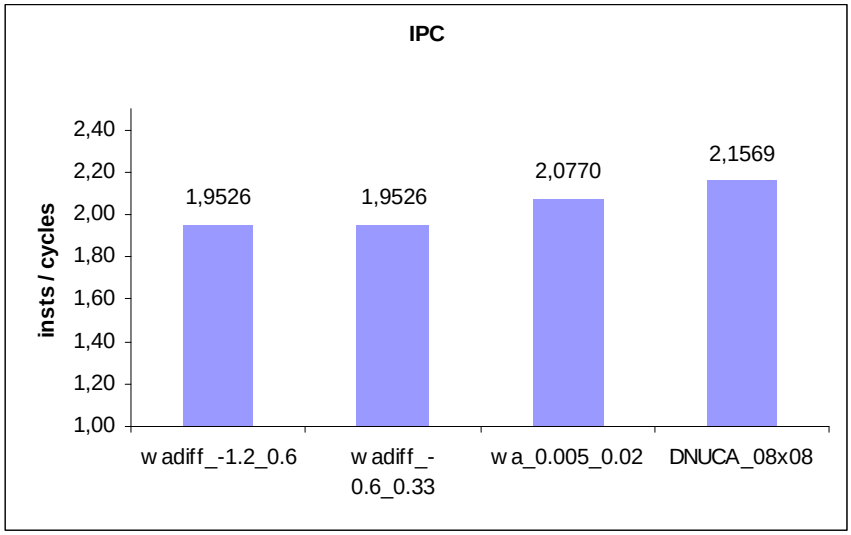


Figura 4.37 - Statistiche IPC relative a *galgel*

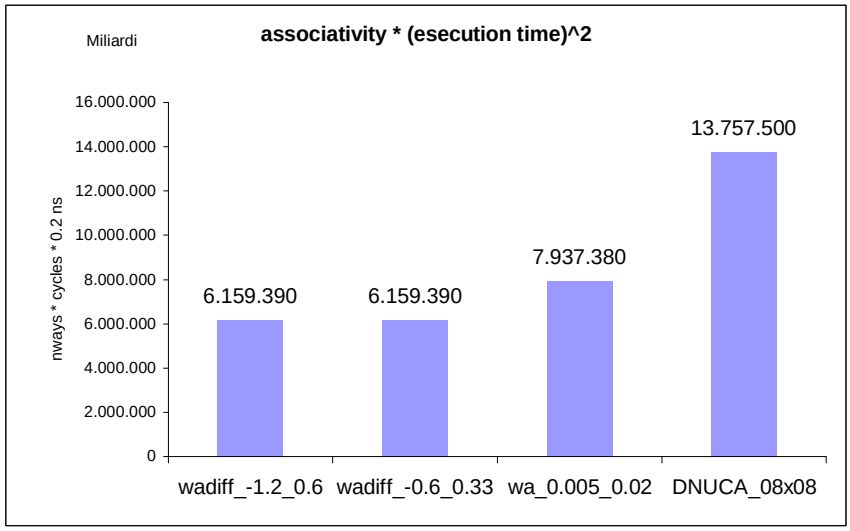


Figura 4.38 - Statistiche $associativity * (\text{esecution time})^2$ relative a *galgel*

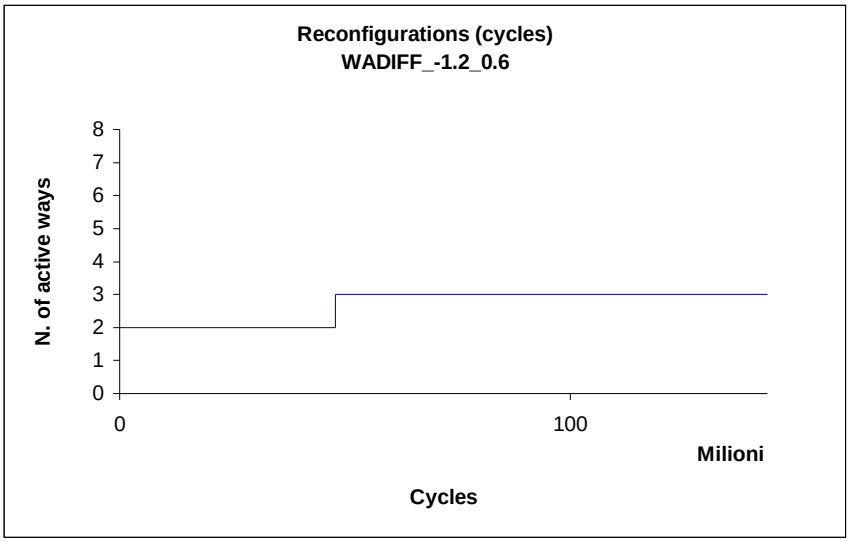


Figura 4.39 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *galgel*

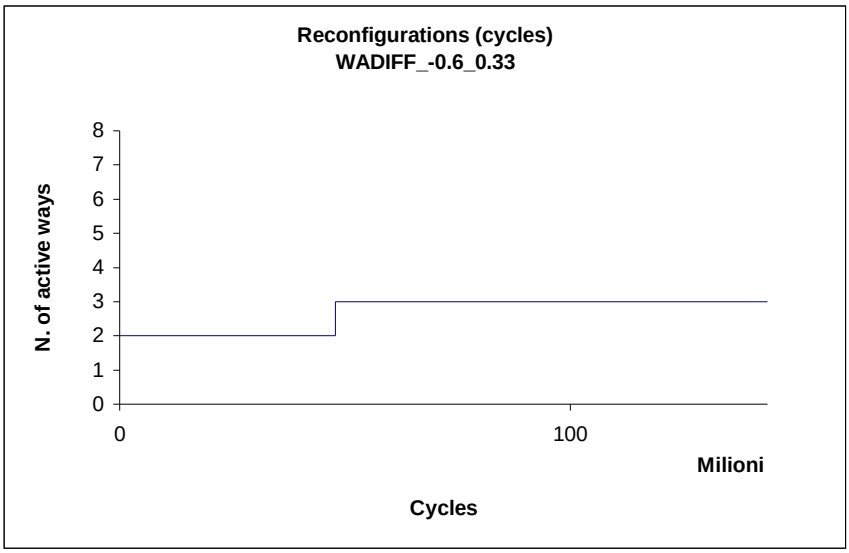


Figura 4.40 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *galgel*

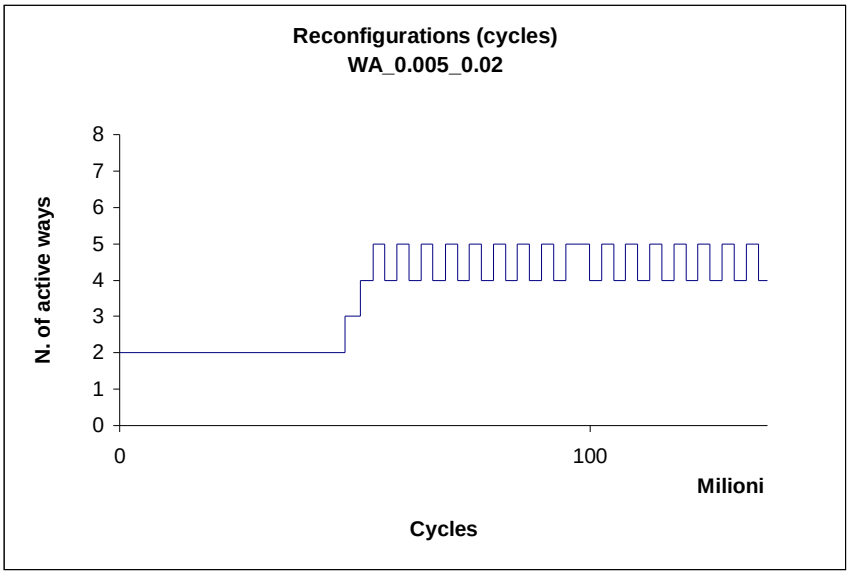


Figura 4.41 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *galgel*

4.9 gcc

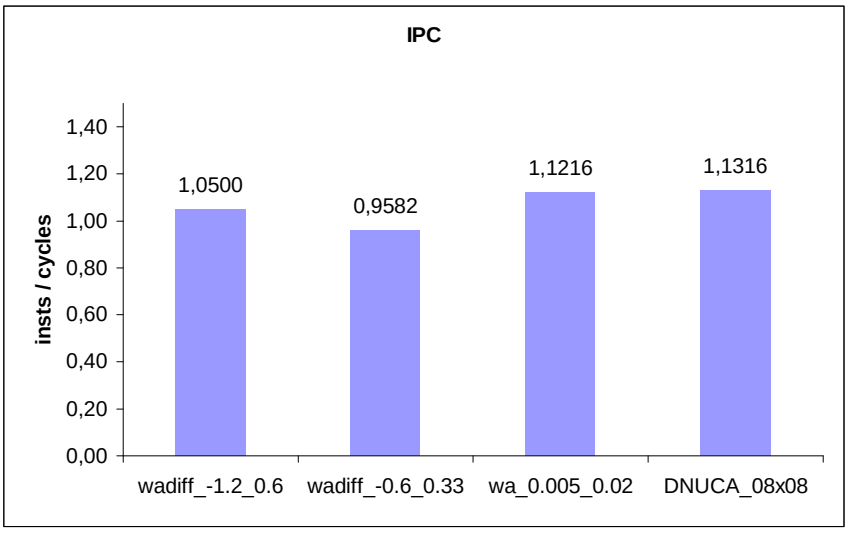


Figura 4.42 - Statistiche IPC relative a *gcc*

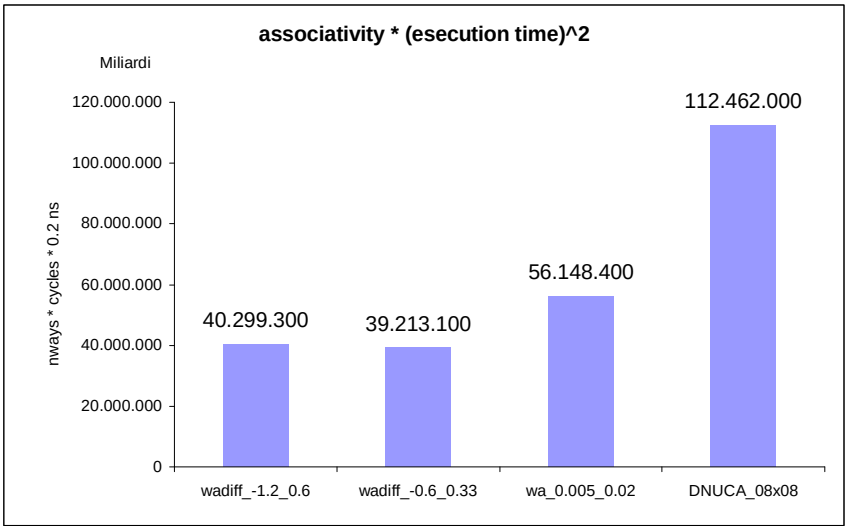


Figura 4.43 - Statistiche $associativity * (\text{esecution time})^2$ relative a *gcc*

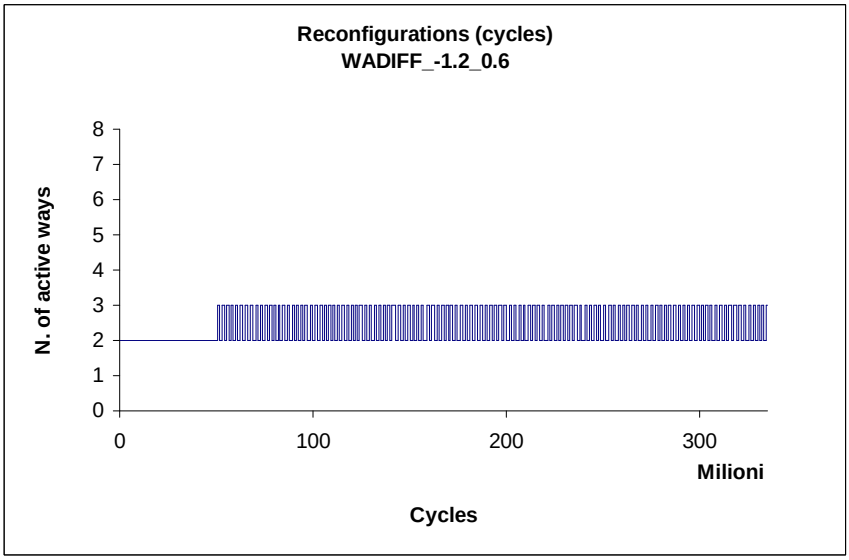


Figura 4.44 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *gcc*

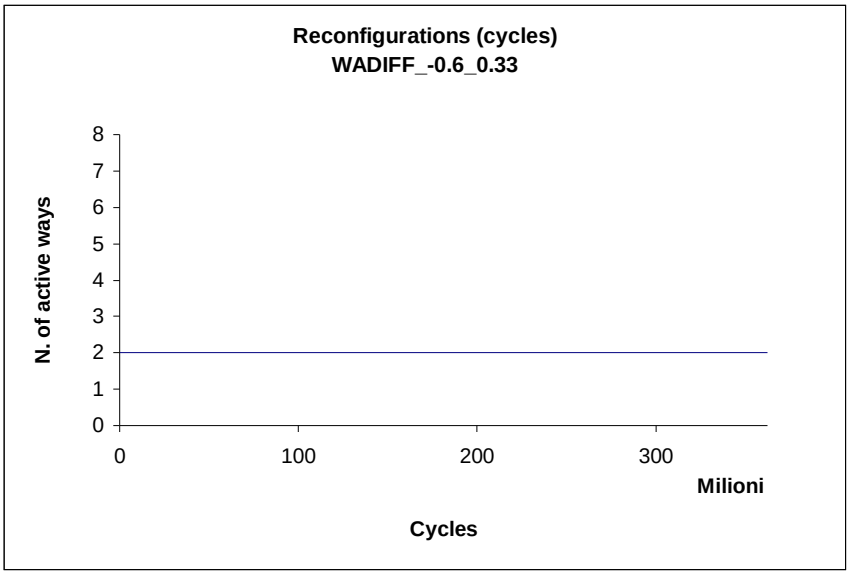


Figura 4.45 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *gcc*

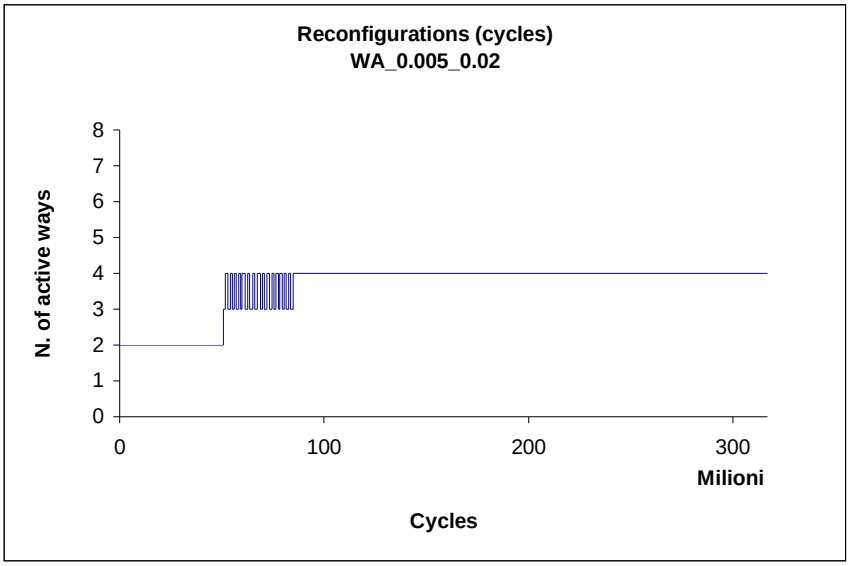


Figura 4.46 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *gcc*

4.10 mcf

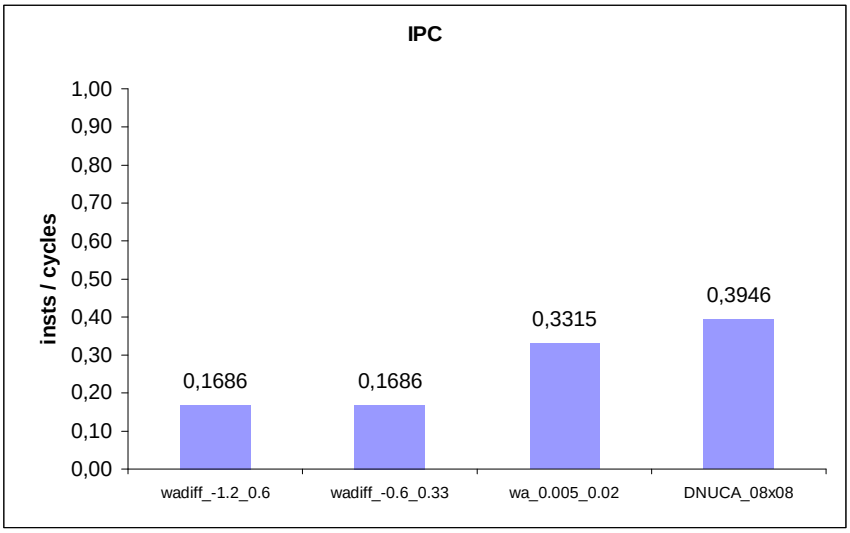


Figura 4.47 - Statistiche IPC relative a *mcf*

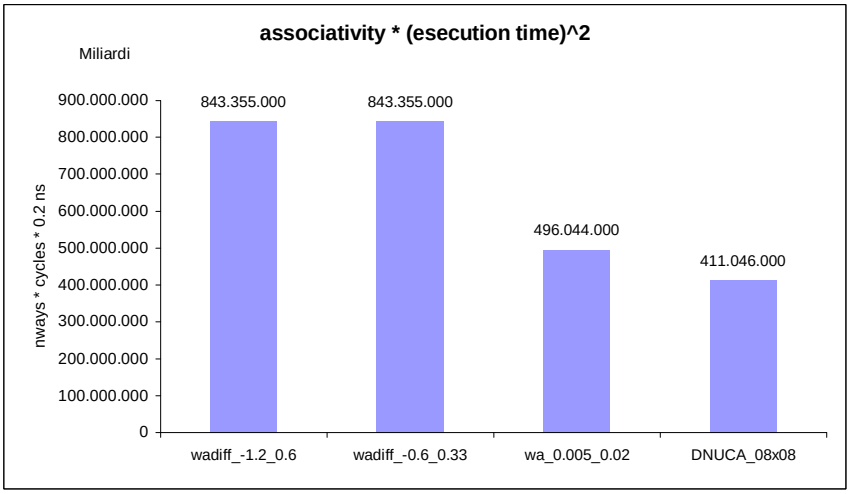


Figura 4.48 - Statistiche $associativity * (\text{esecution time})^2$ relative a *mcf*

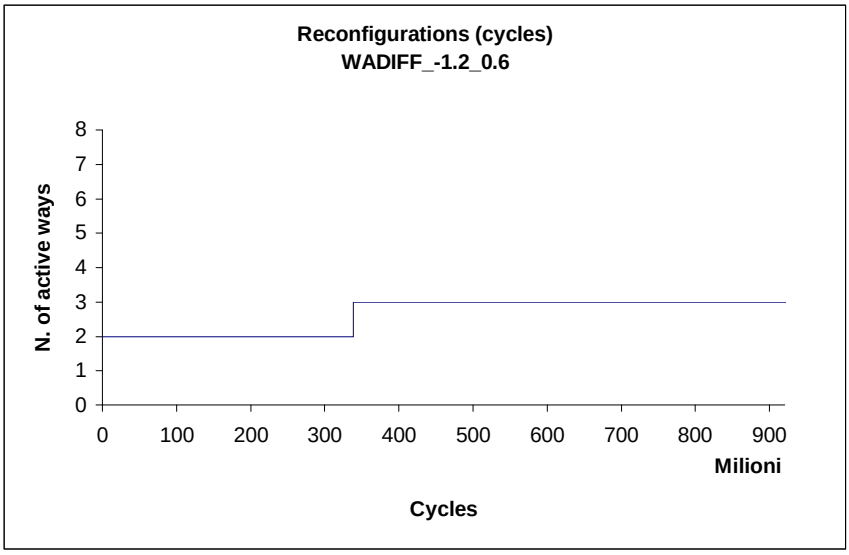


Figura 4.49 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *mcf*

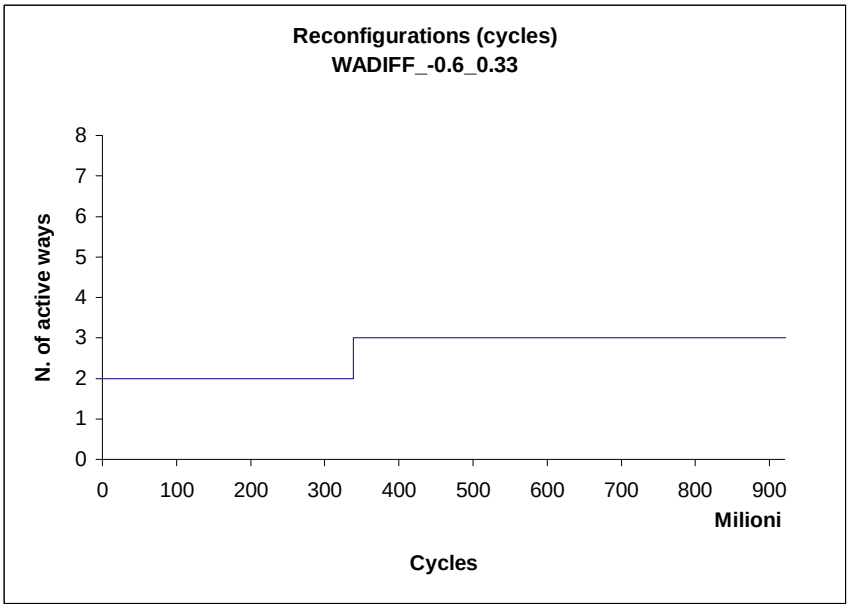


Figura 4.50 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *mcf*

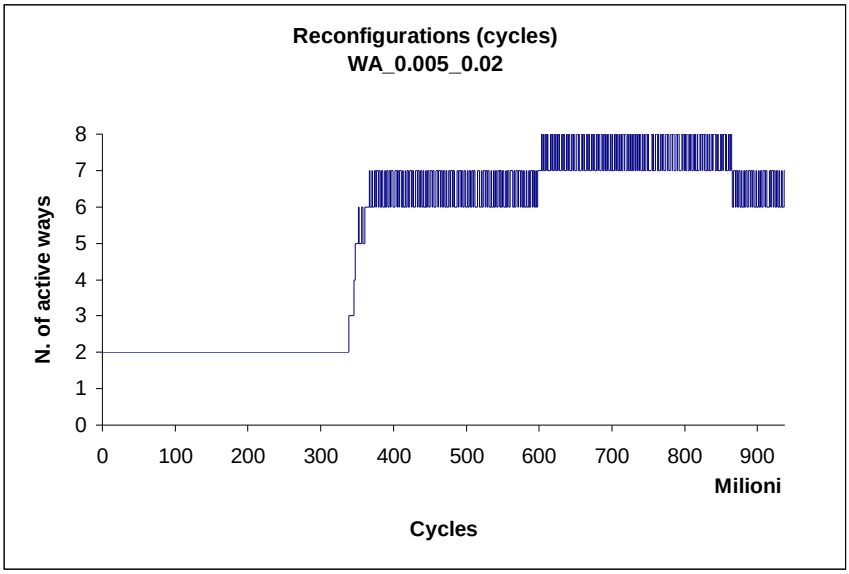


Figura 4.51 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *mcf*

4.11 mesa

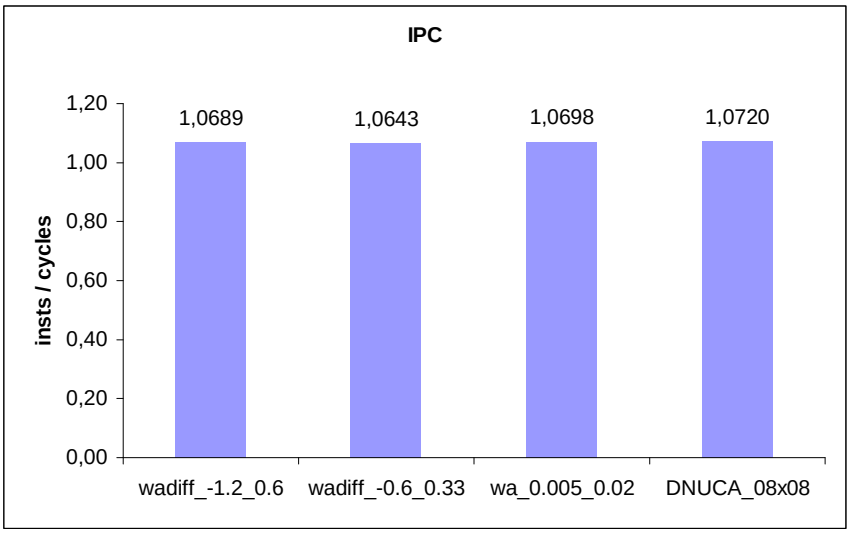


Figura 4.52 - Statistiche IPC relative a *mesa*

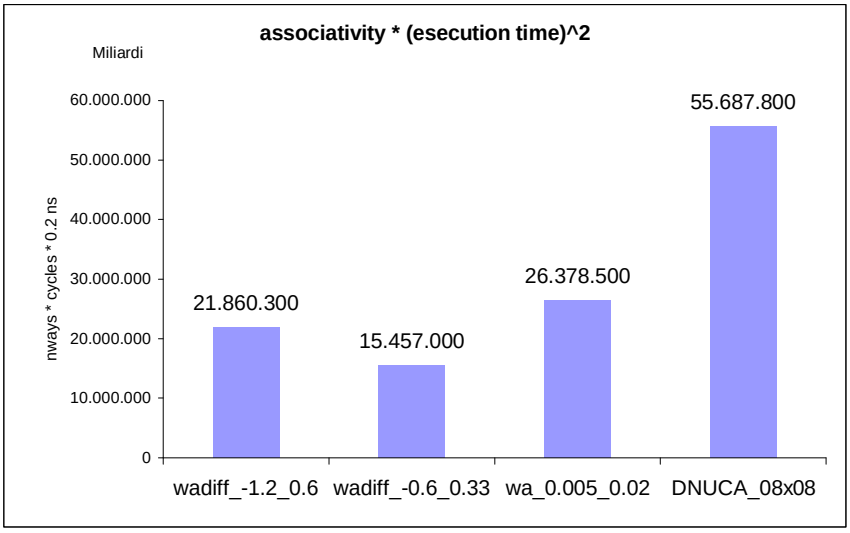


Figura 4.53 - Statistiche $associativity * (\text{esecution time})^2$ relative a *mesa*

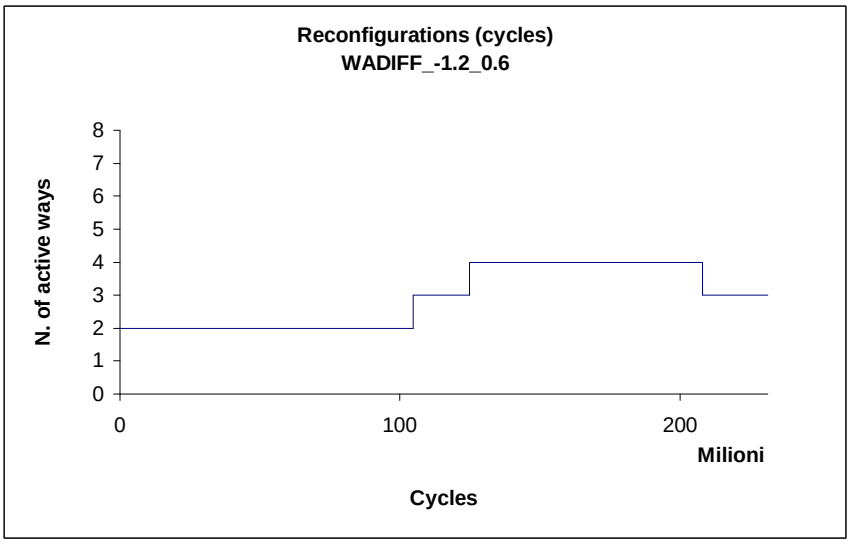


Figura 4.54 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *mesa*

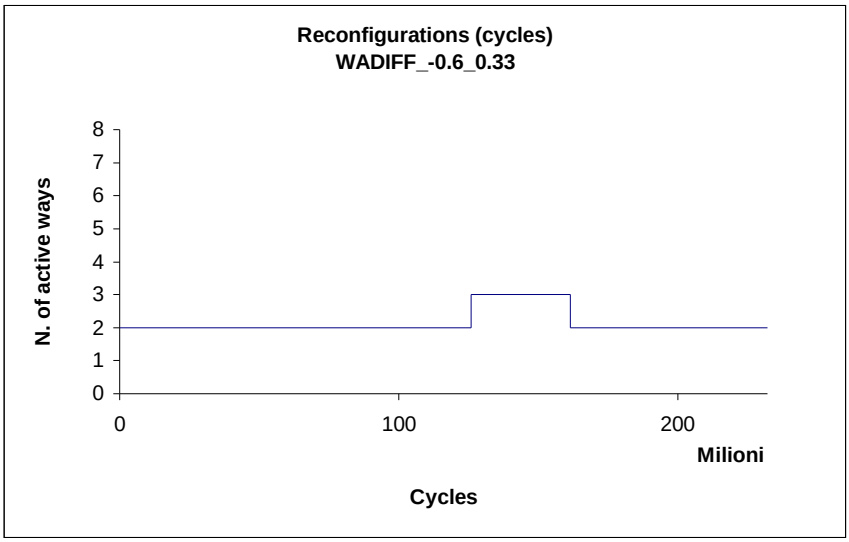


Figura 4.55 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *mesa*

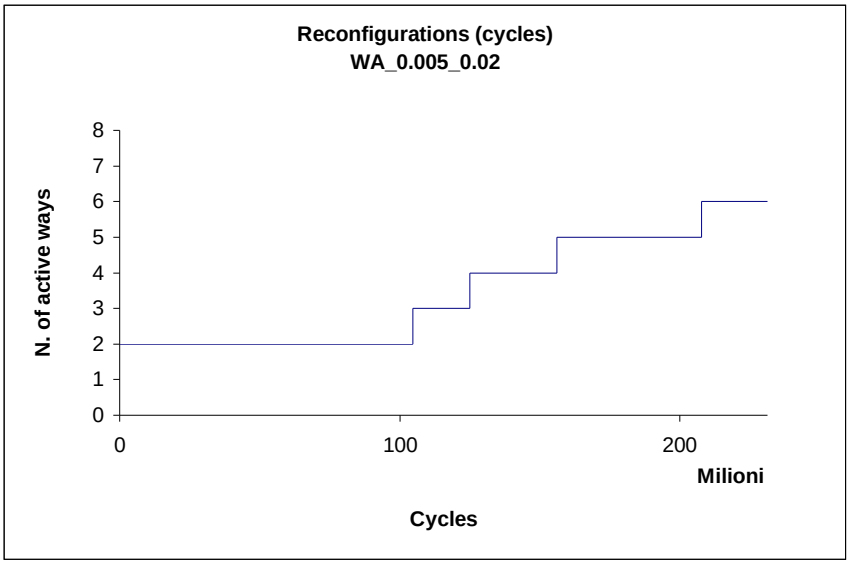


Figura 4.56 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *mesa*

4.12 mgrid

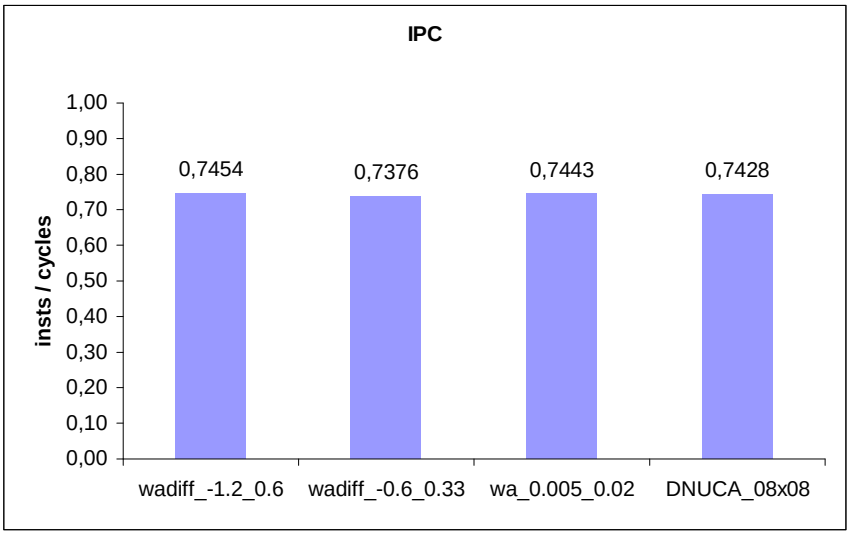


Figura 4.57 - Statistiche IPC relative a *mgrid*

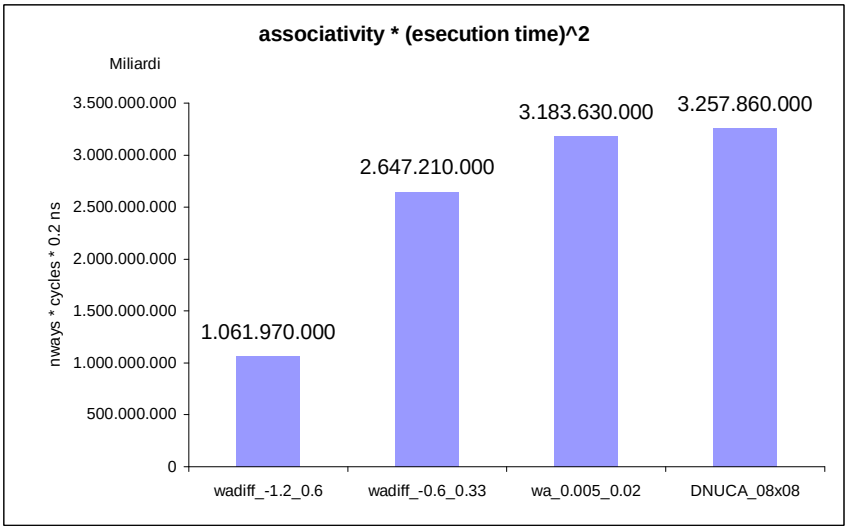


Figura 4.58 - Statistiche $associativity * (\text{esecution time})^2$ relative a *mgrid*

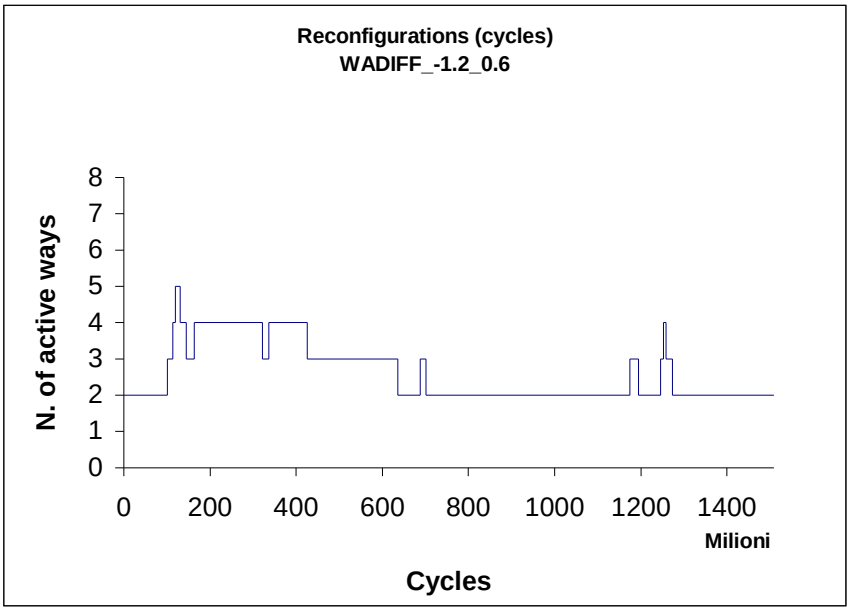


Figura 4.59 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *mgrid*

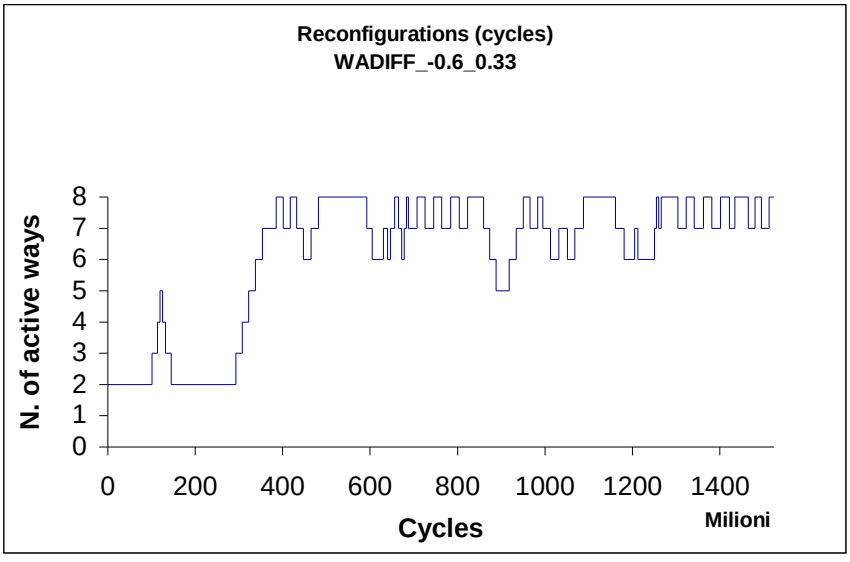


Figura 4.60 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *mgrid*

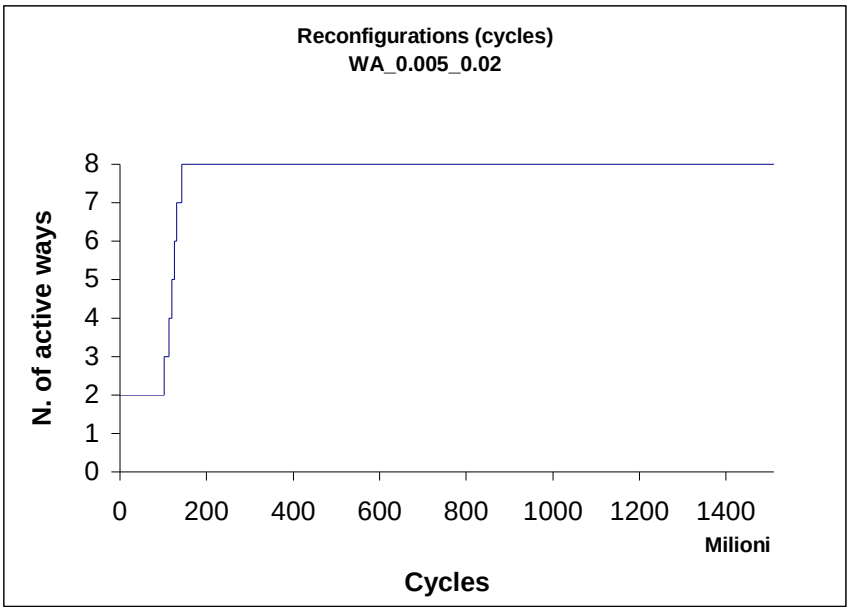


Figura 4.61 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *mgrid*

4.13 parser

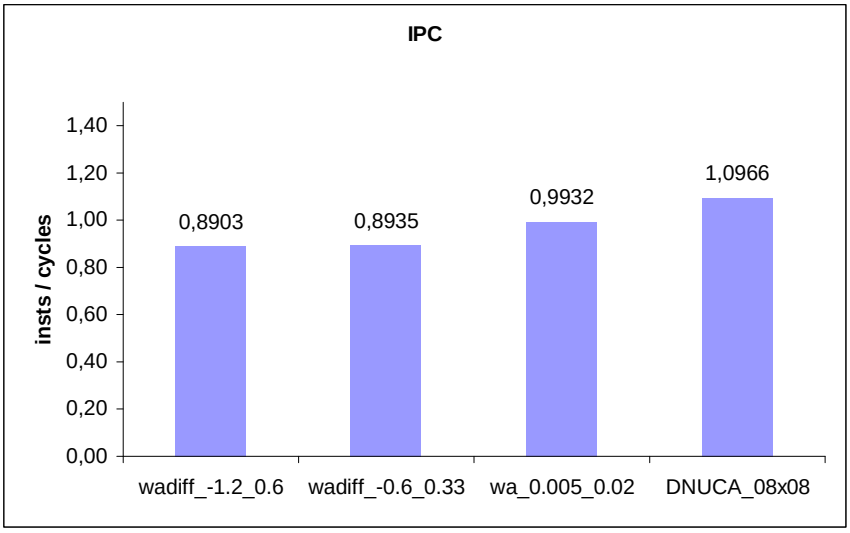


Figura 4.62 - Statistiche IPC relative a *parser*

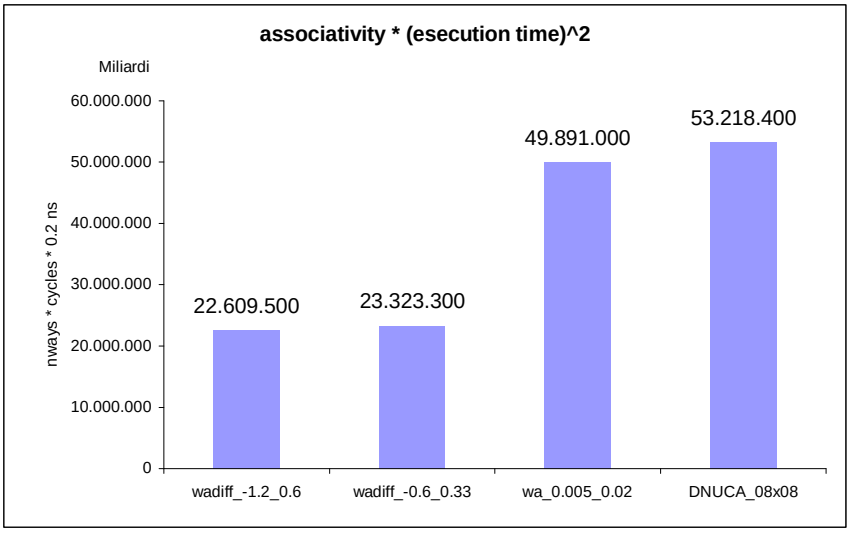


Figura 4.63 - Statistiche $associativity * \left(\text{esecution time}\right)^2$ relative a *parser*

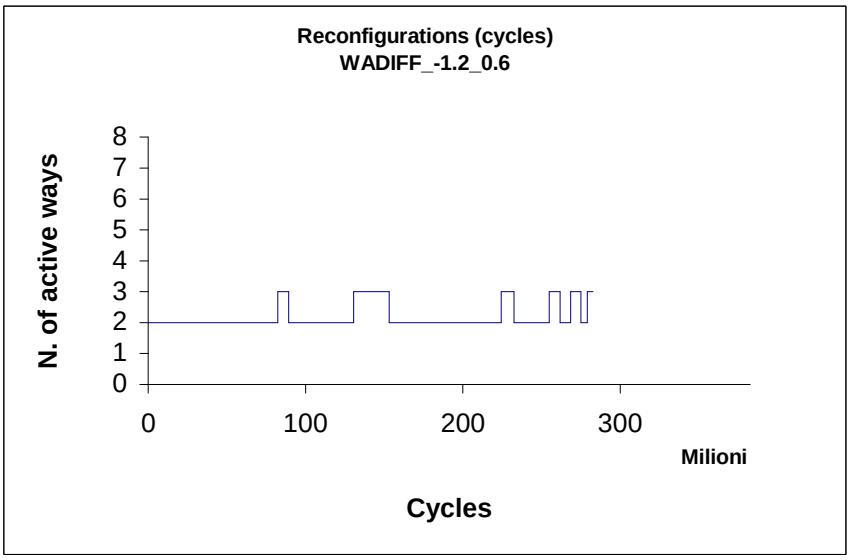


Figura 4.64 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *parser*

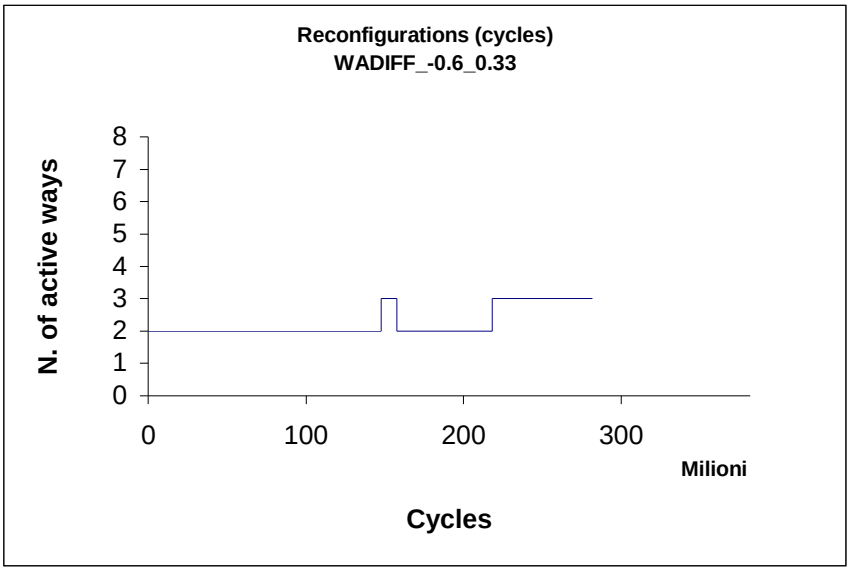


Figura 4.65 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *parser*

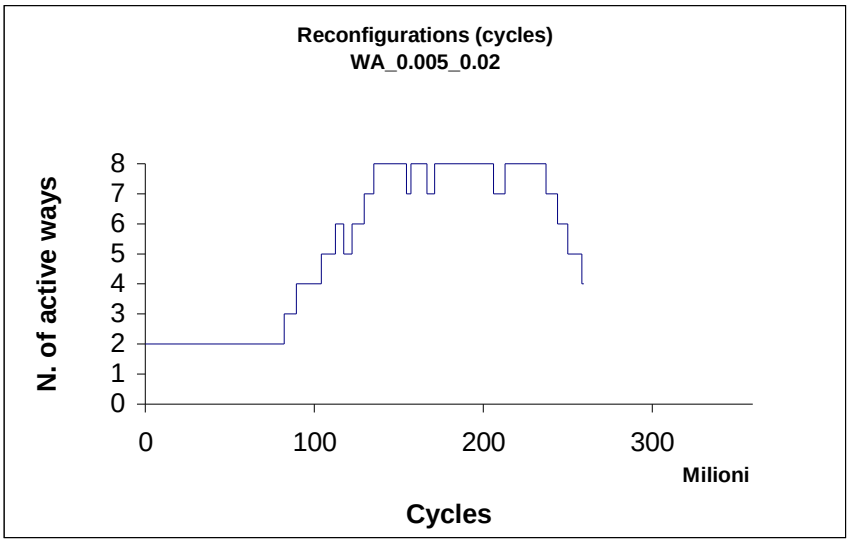


Figura 4.66 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *parser*

4.14 perlbmk

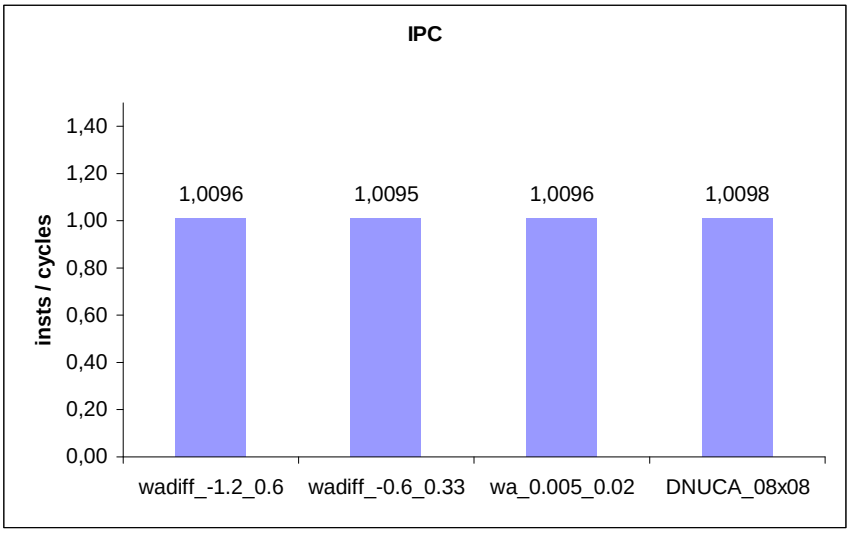


Figura 4.67 - Statistiche IPC relative a *perlbmk*

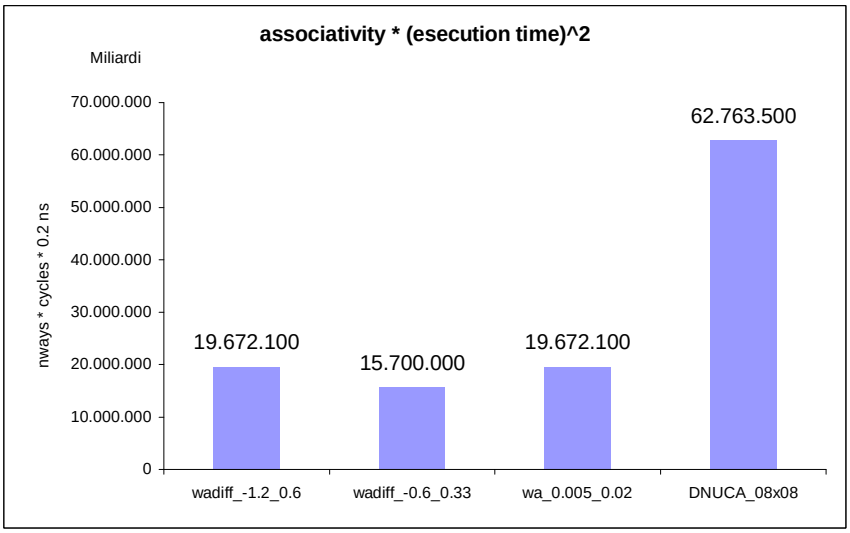


Figura 4.68 - Statistiche $associativity * (\text{esecution time})^2$ relative a *perlbmk*

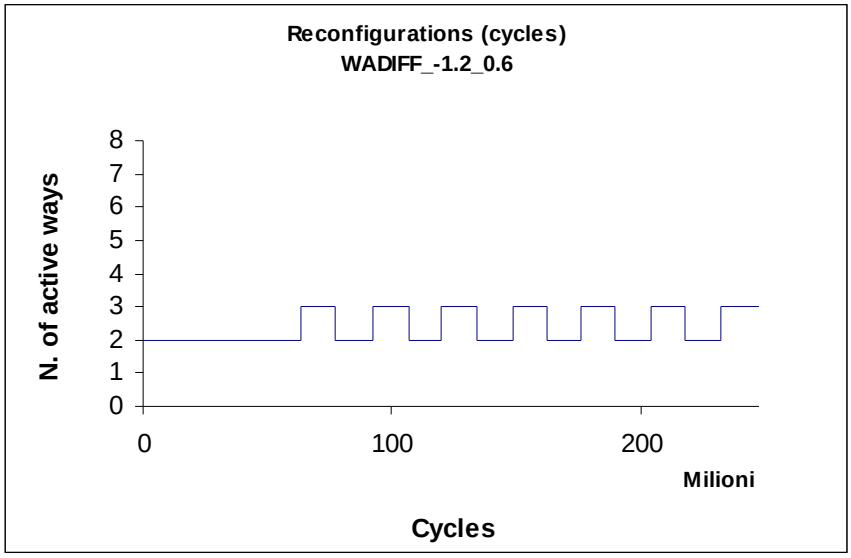


Figura 4.69 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *perlbmk*

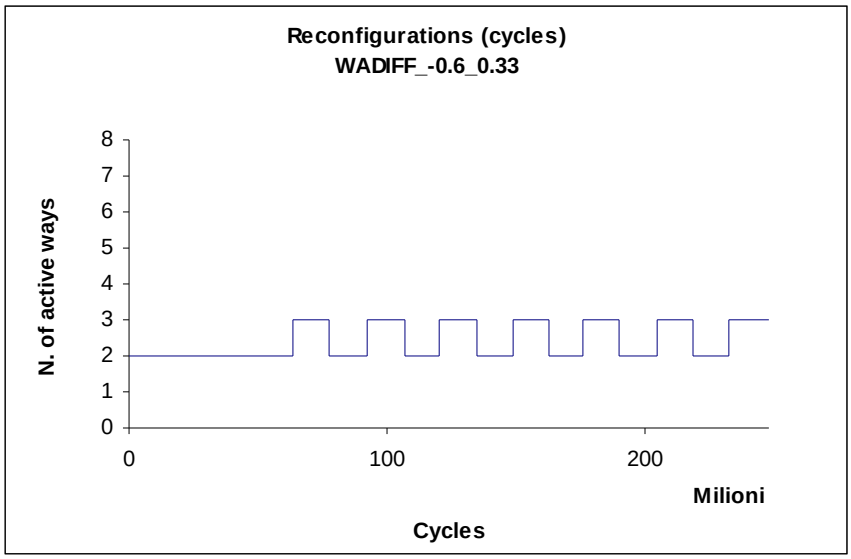


Figura 4.70 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *perlbmk*

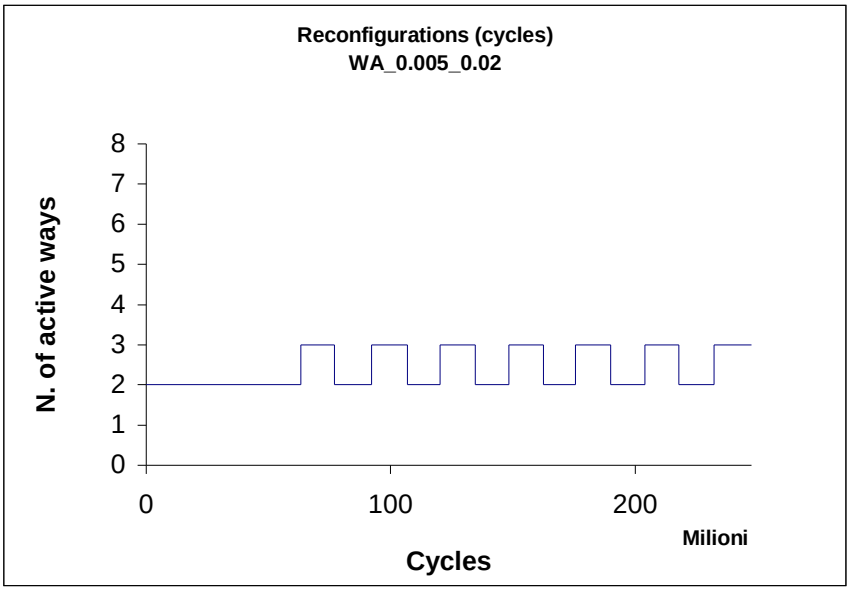


Figura 4.71 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *perlbmk*

4.15 sp

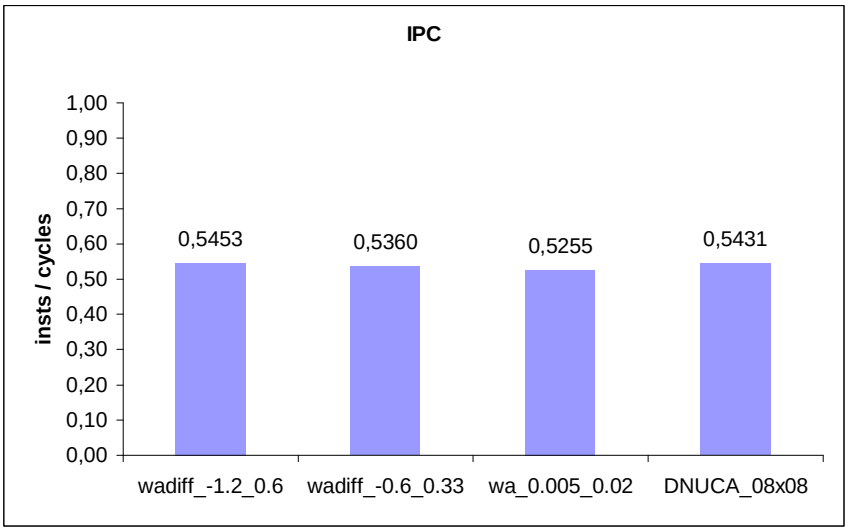


Figura 4.72 - Statistiche IPC relative a *sp*

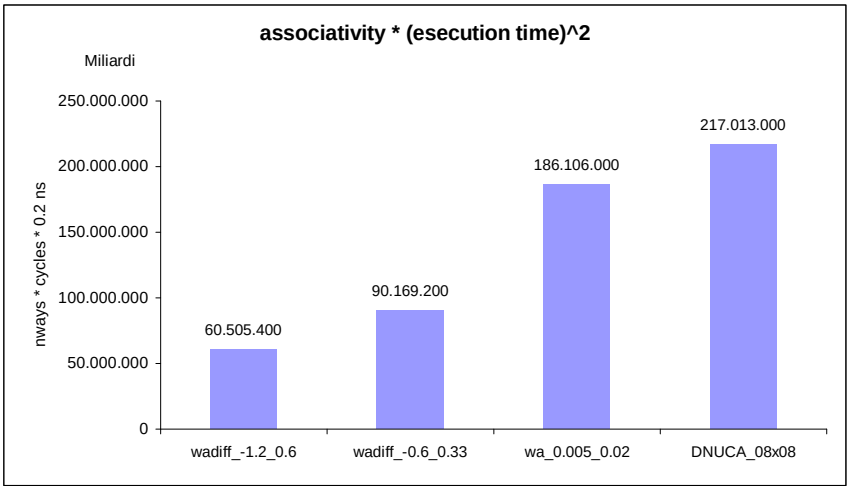


Figura 4.73 - Statistiche $associativity * \left(\text{esecution time}\right)^2$ relative a *sp*

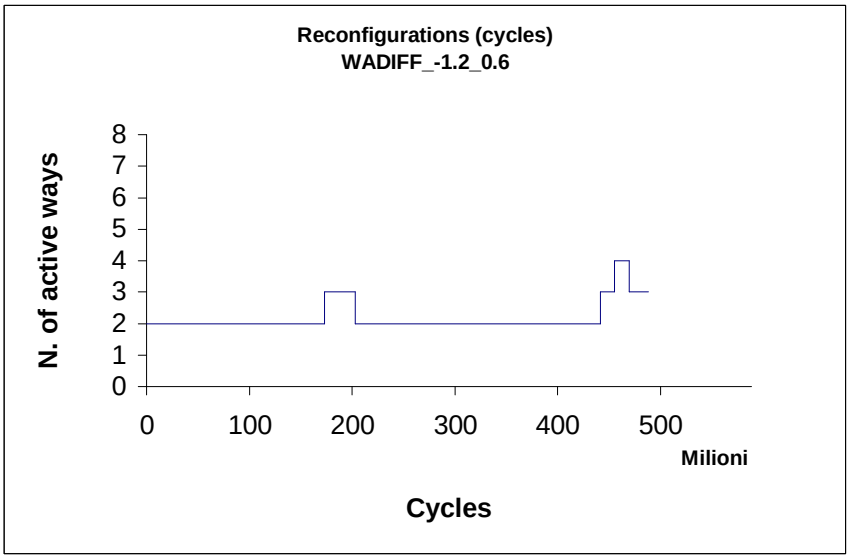


Figura 4.74 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *sp*

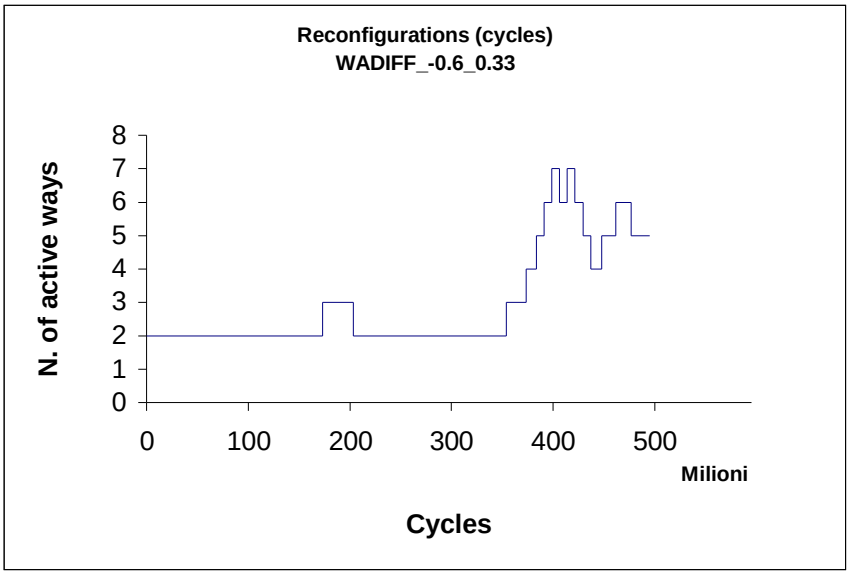


Figura 4.75 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *sp*

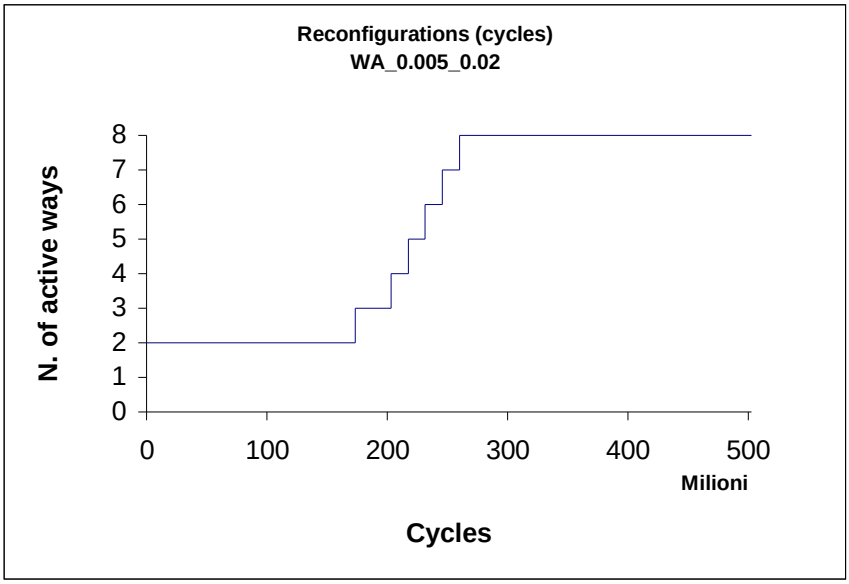


Figura 4.76- Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *sp*

4.16 twolf

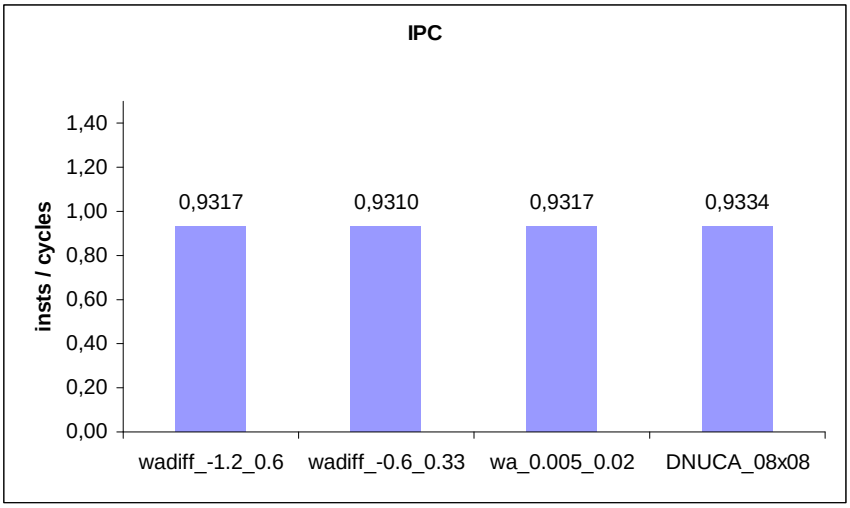


Figura 4.77 - Statistiche IPC relative a *twolf*

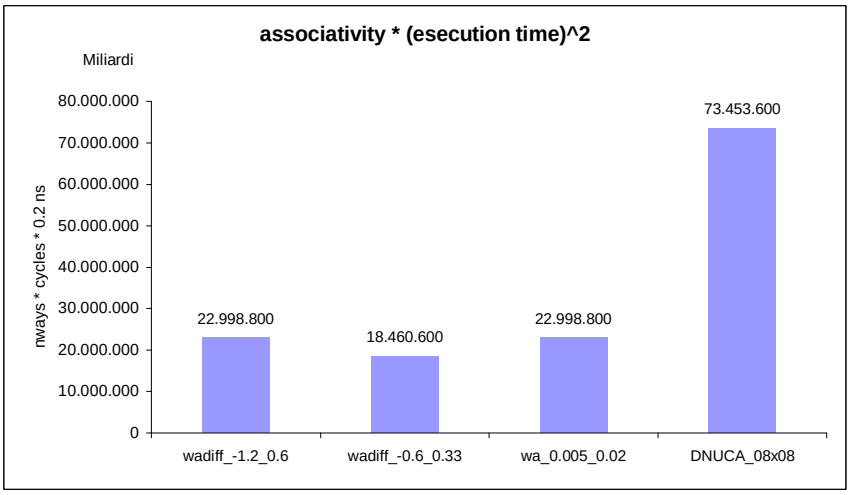


Figura 4.78 - Statistiche $associativity * (\text{esecution time})^2$ relative a *twolf*

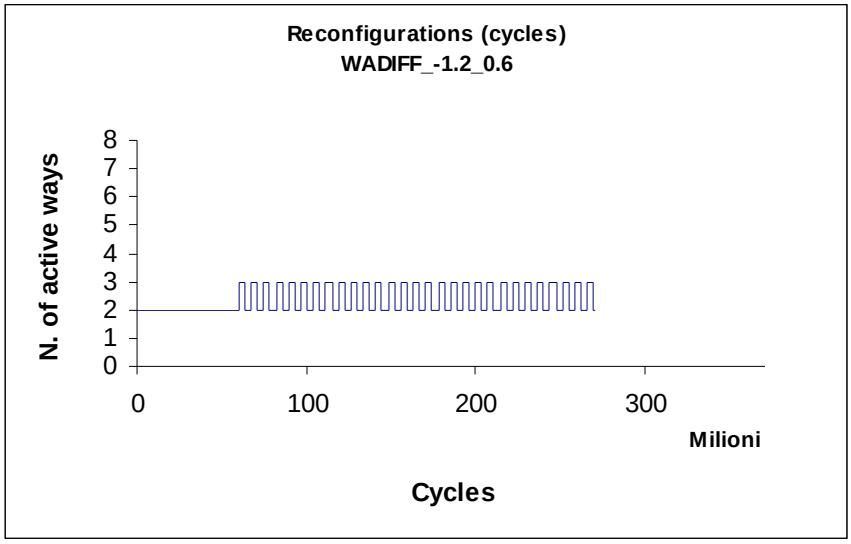


Figura 4.79 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -1.2$ e $T_2 = 0.6$ relative a *twolf*

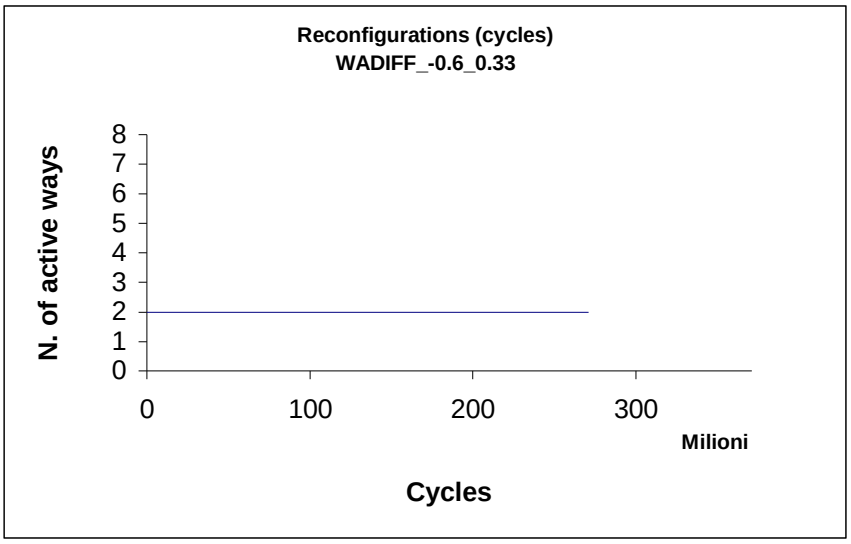


Figura 4.80 - Riconfigurazioni way-adapting differenziale con soglie $T_1 = -0.6$ e $T_2 = 0.33$ relative a *twolf*

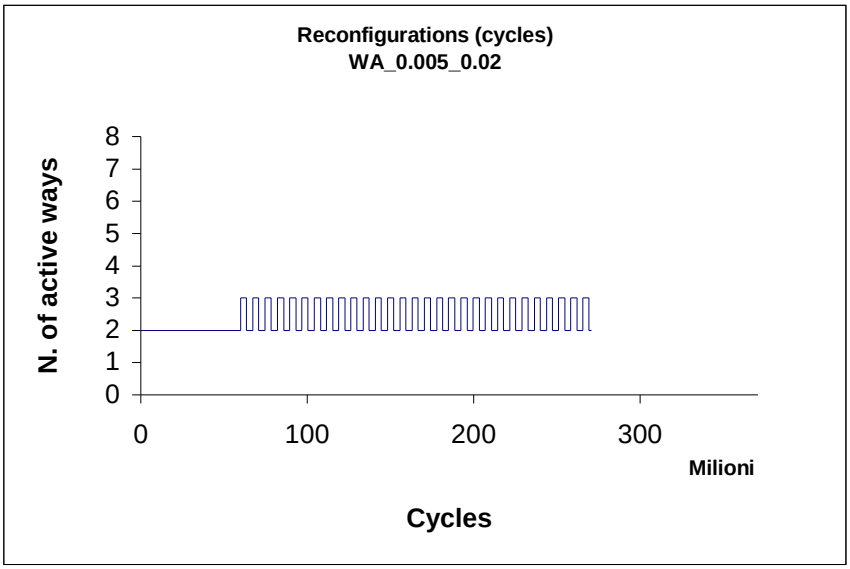


Figura 4.81 - Riconfigurazioni way-adapting normale con soglie $T_1 = 0.005$ e $T_2 = 0.02$ relative a *twolf*

4.17 Risultati finali

In figura 4.83 si riporta l'andamento dell'IPC per tutte le configurazioni di D-NUCA considerate e per tutta la serie dei benchmark.

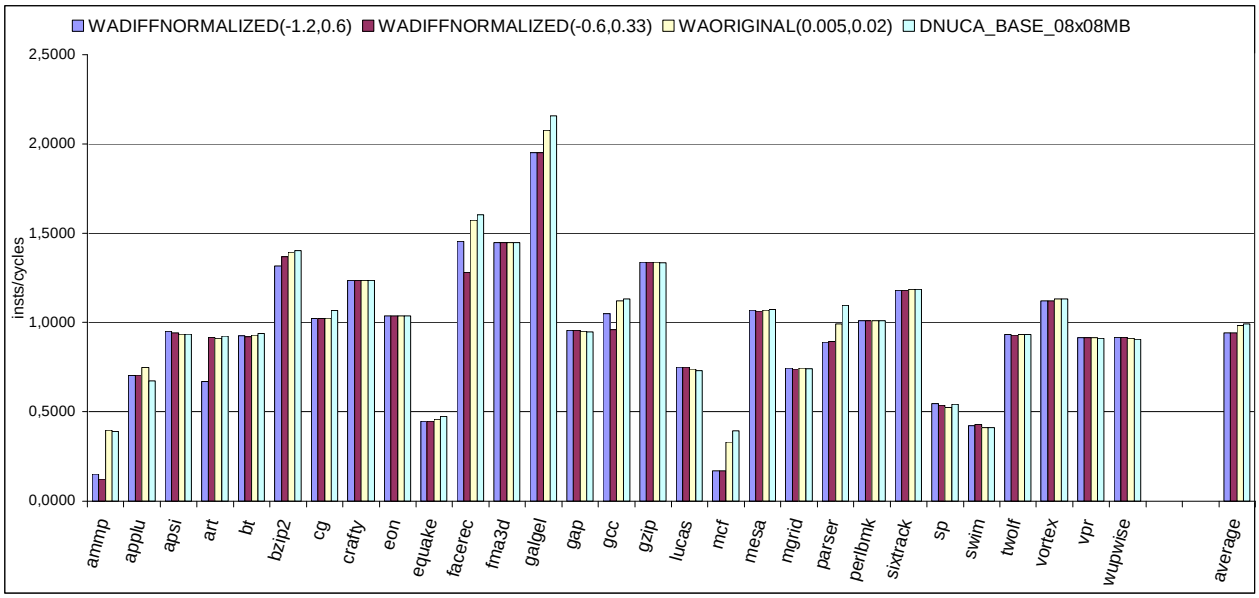


Figura 4.82 - IPC

Nell'average IPC le differenze percentuali, rispetto alla prima configurazione con soglie (-1.2 , 0.6) sono nell'ordine: -0.073 % , +4.22 % e +5.316 %.

La figura 4.84 presenta le associatività medie per tutte le configurazioni di D-NUCA considerate e per tutta la serie dei benchmark.

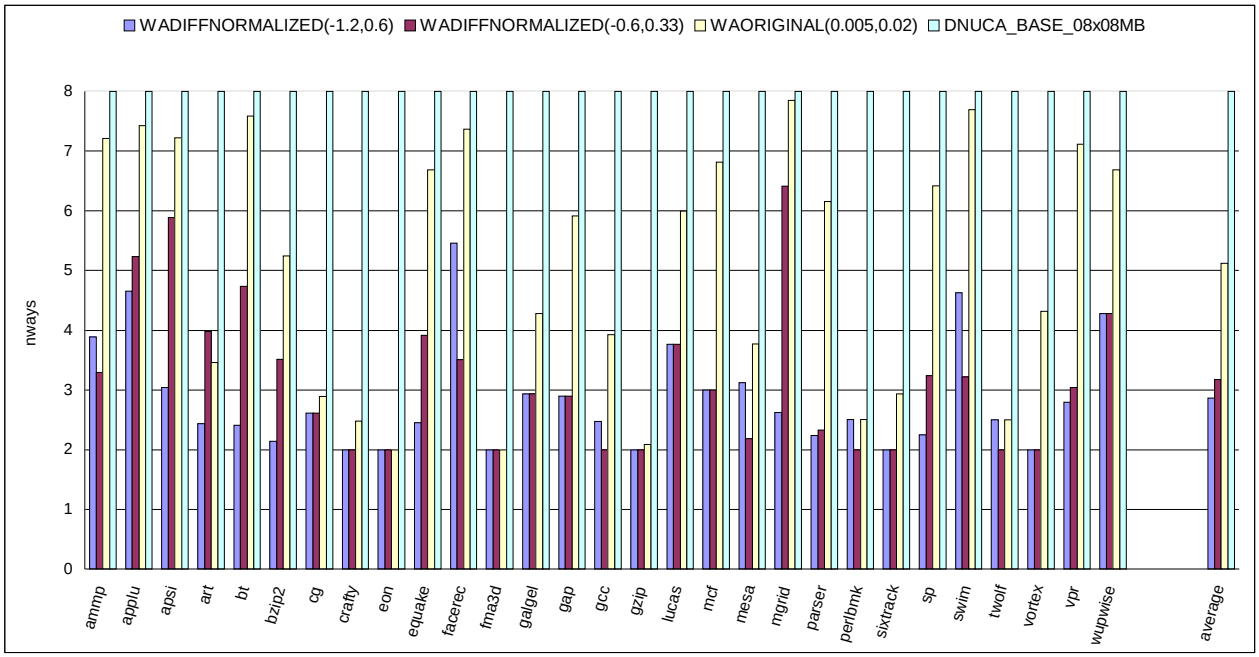


Figura 4.83 - associativity

Nella media sull'associatività le differenze percentuali, rispetto alla prima configurazione con soglie (-1.2 , 0.6) sono nell'ordine: +10.690 % , +78.812 % e +179.235 %.

In figura 4.85 si riportano invece i valori del prodotto tra l'associatività media e il tempo di esecuzione al quadrato per tutte le configurazioni di D-NUCA considerate e per tutta la serie dei benchmark.

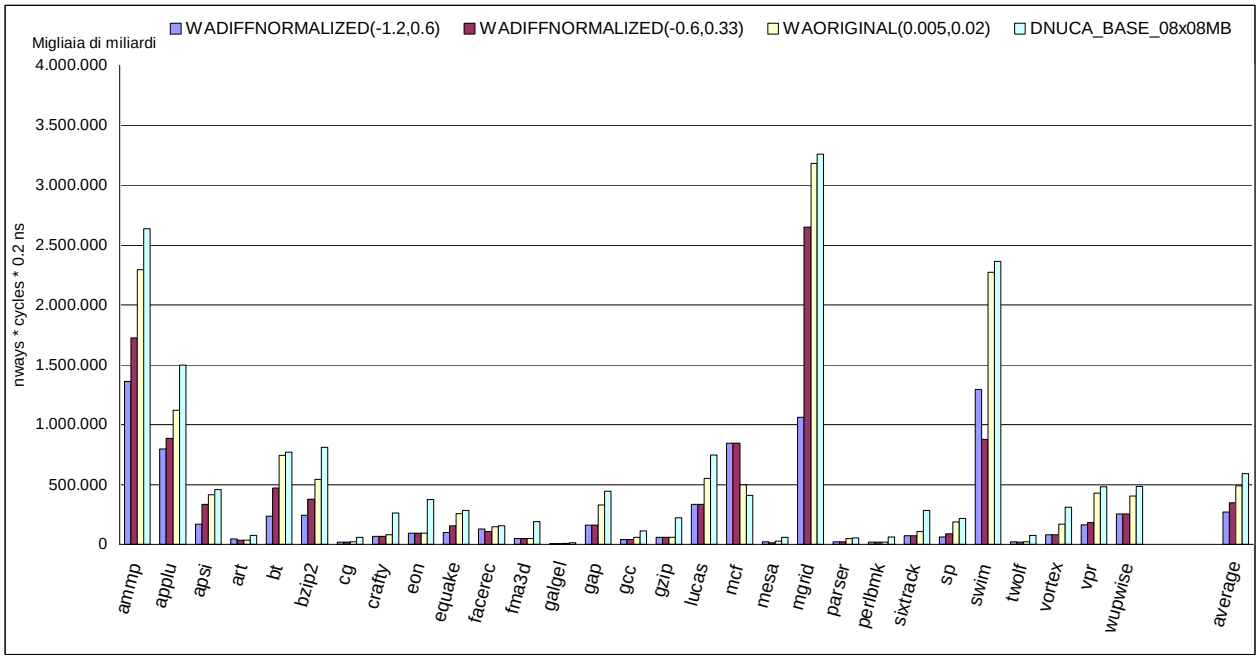


Figura 4.84 - $associativity * (\text{esecution time})^2$

Nell'average sull'energia le differenze percentuali, rispetto alla prima configurazione con soglie (-1.2 , 0.6) sono nell'ordine: +28.234 % , +80.944 % e + 119.478%. Quindi si direbbe che tra le due soglie utilizzate, la coppia che da maggiori vantaggi risulta essere quella con $T_1 = -1.2$ e $T_2 = 0.6$.

In figura 4.86 si riporta l'andamento del miss rate per tutte le configurazioni di D-NUCA considerate e per tutta la serie dei benchmark.

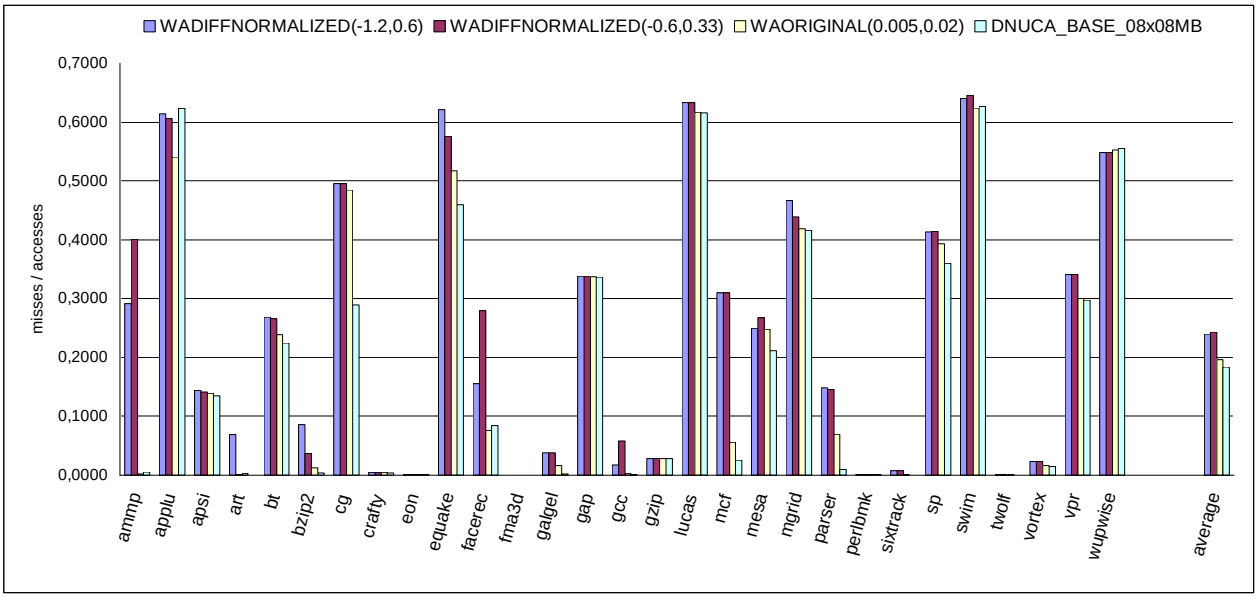


Figura 4.85 – miss rate

Nell'average sul miss rate le differenze percentuali, rispetto alla prima configurazione con soglie (-1.2 , 0.6) sono nell'ordine: +1.356 % , -17.989 % e -23.336 %.

Conclusioni

Il risparmio energetico di una cache D-NUCA può essere migliorato attraverso l'adozione di una tecnica denominata way-adapting, la quale dinamicamente consente di accendere o spegnere porzioni della cache a tempo di esecuzione in base alla località dei riferimenti delle singole applicazioni. Analiticamente la tecnica di way-adapting funziona come segue: se $T_1 < D < T_2$, dove la metrica D rappresenta il grado di località di una cache NUCA , la configurazione corrente della cache è adeguata alla località dell'applicazione e non è necessaria nessuna riconfigurazione; se $D < T_1$ il working set dell'applicazione è contenuto pressoché interamente nella cache e probabilmente c'è la possibilità di ridurre l'associatività senza incorrere in un degrado delle prestazioni: perciò l'azione di riconfigurazione da intraprendere in questo caso è quella di spegnere una via; se $D > T_2$ il working set dell'applicazione richiede una maggiore dimensione della cache e l'azione da intraprendere è invece quella di accendere una via.

Durante il corso della tesi è stato implementato un algoritmo di predizione; la nuova metrica D, che è stata introdotta a ogni passo dell'algoritmo, è uguale alla differenza tra la metrica D di ogni configurazione e la stessa della configurazione successiva; ciò viene ripetuto iterativamente per tutte le configurazioni ammesse dall' applicazione in uso. Il risultato è diviso per la metrica D della configurazione successiva.

In formula: $\frac{D_{new} - D_{old}}{D_{new}}$, dove D è il rapporto tra il numero di hit sull'ultima via (quella più lontana dal controllore L2) e il numero di hit sulla prima via (quella più vicina al controllore L2).

Il lavoro di tesi ha esaminato lo studio di tuning applicato alla tecnica *way-adapting* a cache L2 di tipo D-NUCA. Le soglie che sono state definite e utilizzate nelle simulazioni sono: $T_1, T_2 = (-1.2,\ 0.6)$ e $(-0.6,\ 0.33)$, ricordando $T_1 < T_2$. Sono state fatte anche simulazioni di supporto, considerando le soglie dell'articolo di Kobayashi *et al.*[3], dove per T_1 è stato preso il valore di 0.005 e per T_2 il valore di 0.02.

I benchmark simulati appartengono alle suite NPB e SPEC CPU2000, quest'ultima divisa in due categorie: CINT2000 per le applicazioni con aritmetica intera e CFP2000 per le applicazioni con aritmetica floating-point.

Dalle simulazioni sviluppate nel corso di questo lavoro, prendendo l'average su tutti i benchmark in questione (CINT2000, CFP2000 e NPB), si apprende che sia la pura associatività media che l'associatività media per il tempo di esecuzione, sviluppati con la tecnica differenziale, risultano migliori di quelli riscontrati dai lavori di Kobayashi; questo vale anche confrontando i risultati avuti da una D-NUCA way-adaptable a dispetto di una D-NUCA base (non fa riconfigurazioni).

L'average dell'IPC rimane pressoché su valori vicini in tutte le configurazioni simulate; in particolare è emerso che, per certi benchmark (*apsi*, *gzip* e *sp* su tutti), gli IPC ottenuti con la strategia differenziale sono più alti di quelli ottenuti considerando una D-NUCA base (oppure una D-NUCA way-adaptable ma con le soglie originali definite nell'articolo di riferimento [3]). Discorso completamente opposto per i risultati riguardanti il miss rate, dove in alcuni casi (*equake*, *lucas* e *swim*) si hanno maggiori performance nel caso di una D-NUCA way-adaptable con soglie 0.005 e 0.02.

Quindi i risultati finali hanno dimostrato che, applicando l'algoritmo di predizione studiato nella tesi, su una cache D-NUCA way-adaptable, si ottiene una riduzione del consumo di energia mantenendo le prestazioni pressoché inalterate. Nel esito finale, adottando le soglie

$T_1, T_2 = (-1.2,\ 0.6)$ si ottiene un risparmio del +80.944 % rispetto ad usare la tecnica classica con le soglie di default.

Per gli sviluppi futuri si può estendere la tecnica studiata al caso di multiprocessori, oppure, partendo già dal nostro modello differenziale, di proseguire con la ricerca di configurazioni diverse quali possono essere il numero di vie allo start durante le simulazioni o l'intervallo di pollrate o la fase di warmup, visto che qui, per la maggior parte delle prove, sono state prese sempre le stesse. Inoltre valida alternativa è estendere la piattaforma di valutazione con un modello analitico per il consumo di energia.

Bibliografia

[1] C. Kim, D. Burger, S.W. Keckler: *An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches*, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Ottobre 2002.

[2] A. Desai, B. Mehta, D. Sachdev, G. Muller: *Non uniform cache architectures for wire delay dominated caches*, CS/ECE 752 - Advanced Computer Architecture, 2003.

[3] H. Kobayashi, I. Kotera, H. Takizawa: *Locality analysis to control dynamically way-adaptable caches*. SIGARCH Comput. Archit. News 33, 3 Giugno 2005.

[4] S. Grechi: *Progettazione logica e definizione di protocolli e politiche per le caches D-NUCA triangolari*, tesi di laurea, A.A. 2002/2003.

[5] G. Diodato, R. Tolaro: *Metodologie di progettazione per Dynamic e Triangular-Dynamic NUCA caches*, tesi di laurea, A.A. 2004/2005.

[6] G. Gabrielli: *Applicazione della tecnica di riconfigurazione way-adapting alla progettazione di cache D-NUCA*, tesi di laurea, A.A. 2005/2006.

[7] A. Bardine, P. Foglia, G. Gabrielli, C.A. Prete, P. Stenström: *Improving Power Efficiency of D-NUCA Caches*, Members of the HIPEAC EU Network of Excellence, 2006.

[8] R. Desikan, D. Burger, S.W. Keckler, T.M. Austin: *Sim-alpha: a validated execution driven Alpha 21264 simulator*, Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin, 2001.

[9] D. Burger, A. Kägi, M. S. Hrishikesh: *Memory hierarchy Extensions to the SimpleScalar Tool Set*, Technical Report TR9925, Department of Computer Sciences, University of Texas at Austin, 2001.

[10] Standard Performance Evaluation Corporation, http://www.spec.org.